Pontifícia Universidade Católica
do Rio de Janeiro

**David Beyda**

# Solving the Online Packing IP Under Some Adversarial Inputs

**Dissertação de Mestrado**

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marco Serpa Molinaro

Rio de Janeiro
March 2022

Pontifícia Universidade Católica
do Rio de Janeiro

**David Beyda**

# Solving the Online Packing IP Under Some Adversarial Inputs

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

**Prof. Marco Serpa Molinaro**
Advisor
Departamento de Informática – PUC-Rio

**Prof. Eduardo Sany Laber**
Departamento de Informática – PUC-Rio

**Prof. Marcus Vinicius Soledade Poggi de Aragão**
Departamento de Informática – PUC-Rio

Rio de Janeiro, March the 16th, 2022

**David Beyda**

Majored in Industrial Engineering by the Federal University of Rio de Janeiro (UFRJ), in Brazil. During his Master's in PUC-Rio, worked in the field of online algorithms, primarily towards solving online packing integer programs.



Bibliographic data

Beyda, David

Solving the Online Packing IP Under Some Adversarial Inputs / David Beyda; advisor: Marco Serpa Molinaro. – 2022.

64 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2022.

Inclui bibliografia

1. Informática – Teses. 2. Programas inteiros online. 3. Algoritmos online. 4. Otimização convexa. 5. Aprendizado online. 6. Decisões sequenciais. I. Molinaro, Marco Serpa. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, for their support
and encouragement.

# Acknowledgments

I would like to first thank my advisor, Marco Molinaro. When I met him, as a teacher, his dedication to teaching was evident. He demonstrated both intelectual and personal humility. No question was dumb enough that is didn't deserve an explanation. No question was smart enough, that would go unanswered with fear of wasting other student's time. Then, he became my advisor, and gave me all the time, commitment and guidance I could ask for. For that I am profoundly gratefull.

My family also deserve immensurable gratitude. My Master's was by no means conventional. All the classes were remote, making me spend countless hours in my room, both in classes and during research. Still having spent all this time away from them, they never stopped supporting me, always showing genuine interest in the state of my research, and in my well-being in general. I couldn't have asked for more. Thank you.

Lastly, I'd like to thank the CAPES agency, the brazilian state and its people, for granting me a scholarship for my Master's. I did my best, in effort and in dedication, to ensure those funds were well spent. I hope I have met your expectations. One more time, thank you.

This study was financed in part by the Coordenação de Aperfeiçoaento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

# Abstract

Beyda, David; Molinaro, Marco Serpa (Advisor). **Solving the Online Packing IP Under Some Adversarial Inputs**. Rio de Janeiro, 2022. 64p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

We study online packing integer programs, where the columns arrive one by one. Since optimal algorithms were found for the RANDOMORDER model – where columns arrive in random order – much focus of the area has been on less optimistic models. One of those models is the MIXED model, where some columns are adversarially ordered, while others come in random-order. Very few results are known for packing IPs in the MIXED model, which is the object of our study.

We consider online IPs with $d$ occupation dimensions ($d$ packing constraints), each one with capacity (or right-hand side) $B$. We also assume all items' rewards and occupations to be less or equal to $1$. Our goal is to design an algorithm where the presence of adversarial columns has a limited effect on the algorithm's competitiveness relative to the random-order columns. Thus, we use $\text{OPT}_{Stoch}$ – the offline optimal solution considering only the random-order part of the input – as a benchmark. We present an algorithm that, relative to $\text{OPT}_{Stoch}$, is $(1 - 5\lambda - O(\varepsilon))$-competitive with high probability, where $\lambda$ is the fraction of adversarial columns.

In order to achieve such a guarantee, we make use of a primal-dual algorithm where the decision variables are set by evaluating each item's reward and occupation according to the dual variables of the IP, like other algorithms for the RANDOMORDER model do. However, we can't hope to estimate those dual variables by solving a scaled version of problem, because they could easily be manipulated by an adversary in the MIXED model. Our solution was to use online learning techniques to learn all aspects of the dual variables in an online fashion, as the problem progresses.



## Keywords

Online Integer Programs; Online Algorithms; Convex Optimization; Online Learning; Sequential Decision.

# Resumo

Beyda, David; Molinaro, Marco Serpa. **Resolvendo Online Packing IPs sob a Presença de Entradas Adversárias**. Rio de Janeiro, 2022. 64p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Nesse trabalho, estudamos *online packing integer programs*, cujas colunas são reveladas uma a uma. Já que algoritmos ótimos foram encontrados para o modelo RANDOMORDER– onde a ordem na qual as colunas são reveladas para o algoritmo é aleatória – o foco da área se voltou para modelo menos otimistas. Um desses modelos é o modelo MIXED, no qual algumas colunas são ordenadas de forma adversária, enquanto outras chegam em ordem aleatória. Pouquíssimos resultados são conhecidos para *online packing IPs* no modelo MIXED, que é o objeto do nosso estudo.

Consideramos problemas de *online packing* com $d$ dimensões de ocupação ($d$ restrições de empacotamento), cada uma com capacidade $B$. Assumimos que todas as recompensas e ocupações dos itens estão no intervalo $[0, 1]$. O objetivo do estudo é projetar um algoritmo no qual a presença de alguns itens adversários tenha um efeito limitado na competitividade do algoritmo relativa às colunas de ordem aleatória. Portanto, usamos como *benchmark* $\text{OPT}_{Stoch}$, que é o valor da solução ótima *offline* que considera apenas a parte aleatória da instância. Apresentamos um algoritmo que obtém recompensas de pelo menos $(1 - 5\lambda - O(\varepsilon))\text{OPT}_{Stoch}$ com alta probabilidade, onde $\lambda$ é a fração de colunas em ordem adversária.

Para conseguir tal garantia, projetamos um algoritmo primal-dual onde as decisões são tomadas pelo algoritmo pela avaliação da recompensa e ocupação de cada item, de acordo com as variáveis duais do programa inteiro. Entretanto, diferentemente dos algoritmos primais-duais para o modelo RANDOMORDER, não podemos estimar as variáveis duais pela resolução de um problema reduzido. A causa disso é que, no modelo MIXED, um adversário pode facilmente manipular algumas colunas, para atrapalhar nossa estimação. Para contornar isso, propomos o uso de tecnicas conhecidas de *online learning* para aprender as variáveis duais do problema de forma *online*, conforme o problema progride.

## Palavras-chave

Programas inteiros online; Algoritmos online; Otimização convexa; Aprendizado online; Decisões sequenciais.

# Table of contents

# List of figures

# List of tables

# List of Abreviations

IP – Integer Program

LP – Linear Program

MWU – Multiplicative Weights Update Algorithm

MSMW – Multi-scale Multiplicative Weights Algorithm

VM – Virtual Machine

RHS – Right-hand Side

*For success, like happiness, cannot be pursued; it must ensue, and it only does so as the unintended side effect of one's personal dedication to a cause greater than oneself...*

**Viktor E. Frankl**, *Man's Search for Meaning*.

# 1
# Introduction

Online optimization is a field of Optimization that deals with problems where the inputs are not known upfront. Namely, the input arrives in portions, to which an algorithm must immediately react by making irrevocable decisions, before receiving the next portion of the input. In recent years, research was further motivated by the emergence and relevance of possible applications, such as online display ads allocation and load balancing of VM workloads to physical machines. In this context, the object of our study is the online packing problem, which is an integer program with linear objectives and linear packing restrictions, where each column is revealed only after a decision has been made regarding the last column. We view this problem as follows: a sequence of items is presented to an algorithm, each item has a reward and a $d$ dimensional vector that informs the occupation for each packing constraint. After one item being presented, the algorithm must decide to pick or not pick that item, before seeing the next one. The objective is for the algorithm to maximize the picked rewards, while respecting a capacity of $B$ in each occupation dimension. The simplest example of an online packing problem is the secretary problem: a hidden sequence of $T$ numbers is revealed, one number at a time. The algorithm can pick only one number, and his objective is to maximize the value of the picked number.

When studying algorithms, it's natural to pursue worst-case guarantees. However, sometimes, worst-case guarantees are too pessimistic, in the sense that no good or useful guarantee is possible. In particular, this is true for online packing problems, where the best guarantee possible is $O\left(1/T\right)\text{OPT}$ [4] in the adversarial model. Generally, imposing limits on the adversary can yield more useful guarantees: the adversary prepares every portion of the input, but those portions are presented in random order. This is the RANDOMORDER model, which has taken the spotlight of the area until optimal guarantees were found [1, 14, 19]. Nevertheless, the optimistic RANDOMORDER model is not concerned with "worst-case" time steps, that may happen in practice. This motivated the search for an intermediate model, with both adversarial and random-order time steps, the MIXED model. This model introduces new challenges on how to estimate the dual variables of primal-dual algorithms, since most estimation methods can be manipulated by the adversarial fraction of the instance.

In this work we present a primal-dual algorithm for the online packing problem in the MIXED model: ROBUSTAD (Algorithm 3). Our solution to the problem of estimating the dual variables is to learn those variables as the problem progresses, using online learning mechanisms. In addition, we prove guarantees high probability guarantees for our algorithm, which are relative to the optimum solution $\text{OPT}_{Stoch}$ that considers only the random-order time steps. In other words, the guarantee asserts how much the presence of some adversarial time steps degrades the performance of the algorithm on the RANDOMORDER time steps.

## 1.1 Related Work

The online packing literature traces back to the secretary problem, originated in the realm of *optimal stopping problems*. In the secretary problem, a sequence of $T$ numbers is secretly chosen, randomly shuffled, and then revealed to an algorithm, one number at a time. The algorithm decides when to stop the revelations, and the objective is to stop at the highest number of the sequence. Note this problem is similar to the *online packing problem* in the random-order model, if all items have occupation $1$ and the right-hand side (RHS) capacity is also $1$. The first optimal result for the secretary problem was found in 1960 [13], with probability of success greater than $\frac{1}{e}$ (approaching this value as $n \to \infty$). The algorithm consists of observing the first $(e\,T)$ numbers, registering their maximum, and then stopping at the first number that exceeds the registered maximum.

In 2005, Kleinberg [20] presented an algorithm for a more general problem: the k-secretary problem. It's a variation of the secretary problem where one can pick $k$ numbers and only stop at the $k$-th, with the objective of having the highest number of the sequence in any of the picks. He also studies a variation of this problem where the objective is to maximize the sum of the picked numbers. This last variation is actually an online packing problem where all items have occupation $1$ and the capacity is $k$. Kleinberg presented an algorithm with a competitive ratio of $1 - O\left(1/\sqrt{k}\right)$ for the RANDOMORDER model. He also proved that the competitive ratio achieved by his algorithm is the best possible for this problem. His algorithm is a more complex and sophisticated version of the classic 1960's algorithm.

Now consider the problem where, instead of a number, a set of items is revealed to the algorithm at each time step. Each item has a reward, an occupation vector of $d$ dimensions and a capacity $B$ of occupation in each dimension. At each time step, the algorithm must pick zero or one items from the recently revealed set. The objective is to maximize the sum of items' rewards without exceeding the capacity in any dimension. This problem is the *online packing problem*.

In 2010, Feldman et al. [12] presented a primal-dual algorithm for the online

packing problem in the RANDOMORDER model, achieving a competitive ratio of $1-\varepsilon$, assuming $B \geq \frac{d \log T}{\varepsilon^3}$. Their algorithm observes the first $\varepsilon T$ time steps, than solves the dual problem relative to those first $\varepsilon T$ samples, using capacity $\varepsilon B$ in each dimension. They refer to that problem as the *reduced instance*. This way, they estimate the vector of dual variables $\lambda$, representing the optimum reward/occupation ratio for the reduced instance. Then, for every other time step, the algorithm picks the item that maximizes $(item_{value} - \langle \lambda, item_{occupation} \rangle)$ (not picking any item is always an option). Notice his algorithm solves a LP only once.

Almost concurrently, at the end of 2009, Agrawal et al. [2] made public their work of the DPA (Dynamic Pricing Algorithm), similar to Feldman [12], but solving the LP for the dual variables multiple times, in time steps $\varepsilon T, 2\varepsilon T, 4\varepsilon T, ....$ With that, the assumption on $B$ improved to $B \geq \Omega(\frac{d \log(T/\varepsilon)}{\varepsilon^2})$, but solving multiple LPs required more computational work. Additionally, they proved that under the RANDOMORDER model, no algorithm could achieve a competitive ratio of $1-\varepsilon$ with an assumption of $B$ weaker than $B \geq O\left(\frac{\log d}{\varepsilon^2}\right)$, revealing the state of the art was getting close to the theoretical limits of the problem. Their work was published later, in 2014.

On the same problem, in 2014, Molinaro and Ravi [26] were able to modify the DPA, achieving a competitive ratio of $1-\varepsilon$, with the knapsack size assumption that $B \geq \Omega(\frac{d^2 \log(d/\varepsilon)}{\varepsilon^2})$. Note that they were able to remove the dependence on $T$ for cases where the number of constraints $d$ was greater than 1. Before, only instances with $d = 1$ had results free of any dependency on $T$.

Finally, in 2014, Kesselheim et al. [19] presented the first algorithm to achieve the optimal bound on $B$ for the random-order model: $B \geq \Omega(\frac{\log d}{\varepsilon^2})$. Their approach was to solve, at each time step $t$, a scaled LP of the items already revealed, with capacity $\frac{t}{T}B$. Note this LP allows the solution to contain fractions of items. Then, from the solution of the LP, the fraction of the items revealed in time step $t$ are viewed as probabilities, and the algorithm picks one item randomly, according to those probabilities. Still, this algorithm required a LP to be solved for each time step, which motivated the search for computationally cheaper algorithms.

Finally, Gupta and Molinaro [14] showed how to solve an online packing problem by solving an associated load balancing problem. They show simple algorithms to solve the load balancing problem, and build them into more complex algorithms to solve the online packing problem. They present a high probability guarantee, requiring that $B \geq \Omega\left(\frac{\log d}{\varepsilon^2}\right)$ and assuming items' rewards are small compared to OPT. Then, they remove this last assumption, and present a guarantee with dependency $B \geq \Omega\left(\frac{\log(d/\varepsilon)}{\varepsilon^2}\right)$.

Lastly, Agrawal and Devanur [1] also provided a primal-dual algorithm with optimal guarantees for online packing in the RANDOMORDER model. They show

a connection between online learning and convex optimization, based on the use of the *Fenchel duality*. For the online packing problem, they present an algorithm which tries to obtain the optimal solution by learning the Langrange multipliers $\lambda$ which is factored into $\lambda = Z\theta$. $Z$ is a scalar denoting the scale of the multipliers, and is estimated by observing the first $\delta$ time steps of the problem. $\theta$ such that $\sum_{j\in[d]} \theta_j \leq 1$ is a vector giving the "direction" of the Lagrange multipliers, and is learned over time using online learning mechanisms (prediction with experts).

Regarding the MIXED model, that considers both advesarial and random-order time steps, algorithms with good guarantees have been on the focus of research for some related problems. It has been a topic of interest in the following online problems: secretary problems [8, 17, 18], load balancing [11, 25], welfare maximization [21], facility location [23] and budgeted allocation [24].

## 1.2 Our Results

We present the first algorithm, to our knowledge, that solves online packing IPs in the MIXED model. The main result of this work is algorithm ROBUSTAD (Algorithm 3) and its guarantee (Theorem 5.1), which is presented bellow informally:

**Informal Theorem 1.1.** *Assume $\varepsilon \in (0, 0.1)$ and assume the fraction of adversarial times $\lambda \leq 0.5$. Suppose that each item reward $c_t \in [0, \frac{\text{OPT}_{Stoch}}{B}]$ and the capacity $B \geq \Omega(\frac{(\log\log T + \log(1/\varepsilon))\log d}{\varepsilon^4})$. Also suppose we have a $\text{poly}(n)$ estimate of $\text{OPT}_{Stoch}$. Then, if $B \geq \Omega(\frac{1}{\varepsilon^2}\log[\frac{(\log\log T - \log\varepsilon)}{\varepsilon^2}\frac{d}{\delta'}])$, with probability at least $1-\delta'$, algorithm* ROBUSTAD *achieves*

$$\text{ROBUSTAD} \geq (1 - 5\lambda - O(\varepsilon))\text{OPT}_{Stoch}$$

ROBUSTAD draws inspiration from primal-dual algorithms, in the sense that they use dual variables to evaluate the "constraints-adjusted" rewards of picking an item. However, it does not require the solving of any LPs. Items are accepted if they have a positive gain $item_{value} - Z\langle\theta_t, item_{occupation}\rangle$, where $Z$ is a scalar value updated from time to time (using MSMW algorithm [9]), and $\theta_t$ is a vector balancing the occupation dimensions, updated at each time step (using MWU [3]). Notice that learning $Z\,\theta$ can be interpreted as learning the scale and the direction of Lagrange multipliers. Notice also that our algorithm assumes knowledge of a $\text{poly}(n)$ estimate of OPT, as did some other algorithms in this area (see [14] for a more detailed discussion about this assumption).

Additionally, on the RANDOMORDER model ($\lambda = 0$), we obtain the optimal result of $(1-\varepsilon)$OPT, but we require stronger assumptions on $B$ (by a factor of $\frac{\log\log T + \log(1/\varepsilon)}{\varepsilon^2}$) than other algorithms designed specifically for that model

[1, 14, 19]. We also require an estimate of OPT, which is not a problem in the RANDOMORDER model. Such an estimate can be obtained, for example, by watching a small fraction of the time steps, as done in [1].

Lastly, on section 6, we present experimental results comparing ROBUSTAD and AD on instances generated by the procedure described by Wang [27]. Also, we simulate MIXED instances by applying various types of manipulations on instances generated from the Chu and Beasley [10] procedure. Overall, AD turned out to be more stable and resilient than previously thought, and ROBUSTAD suffered big losses due to the distribution of the capacity into the intervals, and due to the trade-off between learning speed and stability.

# 2
# Online Packing IPs and the MIXED Model

## 2.1
## Online Packing IPs

In the classical online packing problem, a set of items is revealed in each time step, and the algorithm picks up to one item at each time step. For this work, we consider a simpler problem, where only one item is revealed at each time step, as defined below.

**Definition 2.1** (Online Packing IP)**.** *The problem develops during $T$ time time steps. In each time step $t$, an item is revealed. An item has a non-negative reward $c_t \in \mathbb{R}_+$ and an occupation vector $a_t$ with $d$ dimensions and entries in the interval $[0, 1]$. After an item is revealed, the algorithm makes the decision to pick or not the current item, before the next item is revealed. The objective is to maximize the rewards of picked items, while restricting the sum of its occupations to a maximum of $B$ in each dimension. The number of time steps $T$, the capacity $B$ and the occupation dimension $d$ are known upfront.*

That is, the goal is to solve the following packing problem in an online fashion (where the columns come one-by-one):

$$\begin{aligned} \max & \sum_{t=1}^{T} c_t x_t \\ \text{s.t.} & \sum_{t=1}^{T} a_t x_t \leq B\mathbf{1} \\ & x \in \{0,1\}^T, \end{aligned}$$

where $x_t$ denotes the decision of picking or not the item at time $t$.

**Throughout the text, we use $x = \{0,1\}^T$ to denote a solution, and $x_t$ to denote the decision variable in time step $t$. We also use $r_t = c_t x_t$ (which is in $\{0, c_t\}$) to denote the reward obtained at time $t$ by a solution $x$, and $v_t = a_t x_t \in [0,1]^d$ to denote the contribution to the occupation incurred by this time step.**

We note that the assumption of $T$ and $B$ being known upfront can be relaxed: for example, if these quantities are only known within a $(1 \pm \varepsilon)$ factor,

our results will simply incur an additional $(1 - O(\varepsilon))$ loss in the quality of the solution obtained. We also note that we assume that all $d$ constraints have the same capacity/right-hand-side $B$, but this is without loss of generality because we can simply re-scale the rows of the problem appropriately to obtain this property.

## 2.2 Online Input Models

The focus of this work is on Online Packing IPs in the MIXED model. But since it is a mix of the adversarial and the RANDOMORDER models, we first define them and, then, we define the model of interest.

### 2.2.1 Adversarial Model

This model considers that an adversary picks arbitrarily the sequence of items' returns and occupations $(c_t, a_t)$ in advance. The items are presented one-by-one to the algorithm. In particular, the adversary controls the order in which the items are revealed to the algorithm.

### 2.2.2 RANDOMORDER Model

In the RANDOMORDER model (also known as random permutation model), the adversary chooses the set of $T$ items $\{(q_1, u_1), \ldots, (q_T, u_T)\}$ of the instance, but they are sent to the time slots randomly. More precisely, the items $(c_t, a_t)$ are sampled without replacement from the set $\{(q_1, u_1), \ldots, (q_T, u_T)\}$. Then, the sequence of items $(c_t, a_t)_{t\in[T]}$ is presented one-by-one to the algorithm. In short, the adversary looses control over the order in which items are revealed to the algorithm.

Note that as soon as the set of items $\{(q_1, u_1), \ldots, (q_T, u_T)\}$ is chosen, there is an optimal decision associated with each time step. That is, the offline optimum does not depend on the order in which the inputs are presented to the algorithm. In particular, the same holds for the optimal value OPT, which is then a deterministic quantity.

**Definition 2.2.** *(*OPT*) The* ***offline optimum*** *is the optimal solution for the whole instance, namely, the solution of the optimal algorithm that knows the whole instance in hindsight. We use* OPT *to denote the value obtained by this optimal solution.*

### 2.2.3 MIXED Model

We now introduce the MIXED model, which is a mixture of the adversarial and RANDOMORDER models, namely some time steps are "adversarial" and the others are "random-order". More precisely, inputs following this model are built as follows:

1. An adversary partitions the time steps $[T]$ into "adversarial" times $Adv \subseteq [T]$ and "random-order" times $Stoch \subseteq [T]$. This partition is unknown to the algorithm.

2. For the time steps in $Adv$, the adversary chooses its items' rewards and occupations arbitrarily.

3. For the time steps in $Stoch$, the adversary prepares a set of items $\{(q_1, u_1), \ldots, (q_{|Stoch|}, u_{|Stoch|})\}$, but they are sent to the time slots in $Stoch$ randomly. More precisely, the items $(c_t, a_t)$ for the random-order times $t \in Stoch$ are sampled without replacement from the set $\{(q_1, u_1), \ldots, (q_{|Stoch|}, u_{|Stoch|})\}$.

4. The full sequence of items $(c_t, a_t)_{t\in[T]}$ is presented one-by-one to the algorithm.

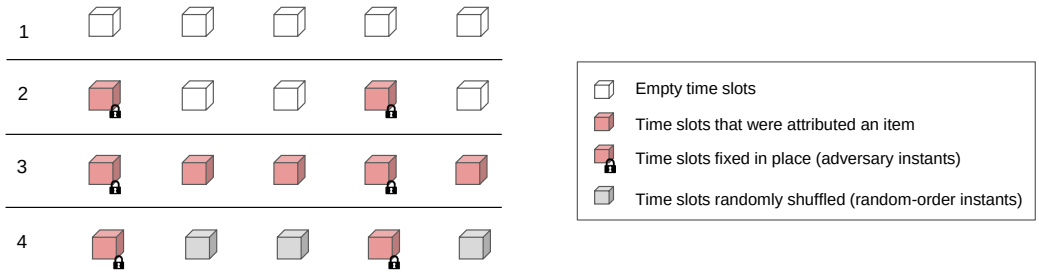


Figure 2.1: Step-by-step building of a MIXED instance

Note that the adversary can make an instance completely adversarial by sending "dummy" random-order items: for example, it can set all random-order items to have value 0. As mentiond above, no non-trivial guarantee is possible in this case. In order to obtain meaningful guarantees, **we compare the algorithm's performance only to the optimum over the random-order times, which we denote by $\mathbf{OPT}_{Stoch}$.** Thus, our goal is to design algorithm for the MIXED model that obtains expected value at least $\alpha\mathrm{OPT}_{Stoch}$ for a constant $\alpha$ as close to 1 as possible.

**Definition 2.3** ($\mathrm{OPT}_{Stoch}$)**.** *We denote by* $\mathrm{OPT}_{Stoch}$ *the value obtained by the offline optimal solution that only considers the* RANDOMORDER *time steps.*

# 3
# Known Results that we Need

In this section we have gathered some ideas and results that we'll need. Sections 3.1 and 3.2 describe some problems and useful results from the online learning area. Online learning algorithms proved to be useful in a variety of situations, from the construction of online portfolios [22], to recommendation systems [16], design of efficient offline algorithms [3], and convex/online/robust/inverse optimization [6, 7, 14, 15]. We will only need two (related) special cases of online learning that we describe in this section.

Additionally, in Section 3.3, we describe an algorithm that is the basis of our analysis.

## 3.1
## Fractional Prediction with Expert Advice

**Definition 3.1** (Fractional Prediction with Expert Advice)**.** *Let us define the simplex* $\triangle^d := \{\theta \in \mathbb{R}^d \ : \ \sum_i \theta_i = 1\}$. *In the* Fractional Prediction with Expert Advice *problem, at time* $t$ *the algorithm needs to choose a vector* $\theta_t \in \triangle^d$. *After that, reward vector* $h_t$ *is revealed. The algorithm gets a reward of* $\langle \theta_t, h_t \rangle$. *This process is repeated for* $T$ *iterations. The objective is to maximize the total reward obtained:* $\sum_{t=1}^{T} \langle \theta_t, h_t \rangle$.

The benchmark used to evaluate algorithms for this setting is the value obtained by choosing the best fixed vector $\theta^* \in \triangle^d$ in hindsight, at every time step: $\sum_{t=1}^{T} \langle \theta^*, h_t \rangle$. Thus, good algorithms should achieve a total reward that satisfy

$$\sum_{t=1}^{T} \langle \theta_t, h_t \rangle \geq \alpha \max_{\theta \in \triangle^n} \sum_{t=1}^{T} \langle \theta, h_t \rangle - \beta,$$

for $\alpha$ as close to 1 as possible, and $\beta$ as close to 0 as possible.

There are known algorithms that achieve those guarantees, such as the Multiplicate Weights Update Algorithm (MWU) [3], described in Appendix A.1. However, the Fractional Prediction with Expert Advice problem we have to solve will be in the domain of a "full-dimensional simplex", defined by $\blacktriangle^d := \{\theta \in \mathbb{R}^d \ : \ \sum_i \theta_i \leq 1\}$. Bellow, we present a result for the MWU in $\blacktriangle^d$, which follows directly from Theorem 5 of [3], and a more extensive demonstration is presented in Appendix A.2.

**Lemma 3.2** (MWU)**.** *Suppose that all reward vectors $h_t$ belong to $[-3,3]^d$. Then there is an algorithm MWU such that for every $\varepsilon \in (0, \frac{1}{2}]$ its decisions satisfy*

$$\sum_{t=1}^{T}\langle\theta_t, h_t\rangle \geq \sum_{t=1}^{T}\langle\theta^*, h_t\rangle - \varepsilon\sum_{t=1}^{T}|\langle\theta^*, h_t\rangle| - \frac{3\log(d+1)}{\varepsilon}$$

*where $\theta^* \in \blacktriangle^d$ is the fixed vector that maximizes the rewards in hindsight.*

## 3.2 (Integral) Prediction with Expert Advice

**Definition 3.3** (Integral Prediction with Experts)**.** *In this problem, there is a set of $n$ "experts". At time $t$, the algorithm needs to choose (possibly randomly) an expert $i_t \in [n]$. After that, it sees the reward $(h_t)_i$ obtained by each expert $i \in [n]$ (so $h_t$ is a reward vector with $n$ coordinates). Again the goal is to maximize the total reward obtained.*

Similarly to the fractional problem, we evaluate algorithms for the integer problem by comparing them to the best expert in hindsight; that is, to obtain

$$\sum_{t=1}^{T}(h_t)_{i_t} \geq \alpha\max_{i\in[n]}\sum_{t=1}^{T}(h_t)_i - \beta,$$

for $\alpha$ as close to 1 as possible, and $\beta$ as close to 0 as possible.

Notice that the problem from the previous section (definition 3.1) can indeed be seen as a "fractional" version of this problem, where the $i$th coordinate $(\theta_t)_i$ says how much the algorithm wants to use of expert $i$, at time $t$.

We will need the following result, which while related to Lemma 3.2, offers a guarantee that scales better with the range of the rewards for the individual experts. The following can be obtained directly from the main result of [9]. The algorithm that holds that result, MSMW, is reproduced in Appendix A.3.

**Lemma 3.4** (MSMW)**.** *Consider experts with non-negative rewards. Let $c \in \mathbb{R}^n$ be a vector that upper bounds the reward vectors of every time $t$, namely $(h_t)_i \leq c_i$ for every $i \in [n]$ and for all $t$. Then there is an algorithm MSMW such that for all $\varepsilon \in (0,1]$, its (random) decisions satisfy*

$$\mathbb{E}\sum_{t=1}^{T}(h_t)_{i_t} \geq (1-\varepsilon)\sum_{t=1}^{T}(h_t)_i - O\left(\frac{c_i\log(n/\varepsilon)}{\varepsilon}\right)$$

*for every expert $i \in [n]$.*

Notice how the guarantee presented on Lemma 3.2 differs from this one, and not only by the randomness and expectations. The former compares the algorithm's

rewards with the rewards obtained by the best fixed decision in hindsight. The latter allows us to compare the algorithm's expected rewards with the rewards obtained by any fixed expert $i$ we want, and not only the best one.

## 3.3 Agrawal-Devanur (AD) Algorithm for Online Stochastic LPs

Agrawal & Devanur presented an online packing algorithm [1] that works for the RANDOMORDER model and is central to this work. We refer to it as Algorithm AD, and present it below. Recall $c_t$ and $a_t$ denote, respectively, the reward and occupation vector of the item presented in time step $t$. Also, $v_t = a_t x_t$ is the capacity occupation incurred by the algorithm relative to the choice $x_t = \{0, 1\}$ in time step $t$. Below, we describe Algorithm AD and show how to set parameter $\delta$.

**Algorithm 1** Algorithm AD for the Random Order Model

**procedure** AD($B$,$T$)
  **for** $t = 1, 2, \ldots, \delta T$ **do** ▷ Observation phase
    Don't pick any item
  Set $Z \leftarrow \frac{\text{OPT}'}{\delta B}$ based on the items seen so far
  Initialize $\theta_{\delta T+1}$ from MWU with $d + 1$ experts and gains in range $[-1, 1]$
  **for** $t = \delta T + 1, \ldots, T$ **do** ▷ Active phase
    Pack item if $c_t - Z\langle \theta_t, a_t \rangle \geq 0$
    **if** any dimension of the occupation exceeds $B$ **then**
      Exit
    Compute the gains $h_{t,j} = v_{t,j} - \frac{B}{T}$ of each dimension $j = 1, \ldots, d$ and set $h_{t,0} = 0$
    Send $h_t$ to the MWU to obtain the next occupation weights $\theta_{t+1}$

| Param. | Value |
|---|---|
| $\delta$ | $12\frac{4\varepsilon^2}{\log d}\log\left(\frac{d+2}{\varepsilon^2}\right)$ |

Table 3.1: Parameter setting for Algorithm AD.

AD is a primal-dual algorithm that tries to learn the Lagrange multipliers that best insert the constraints into the objective function. This is done by separating the scale $Z$ of the multipliers from their direction vector $\theta$, and, at each time step $t$, the decision is made trying to maximize the new objective function, that includes the constraints.

To give another view of the algorithm, notice it works by attributing weights $\theta_t$ to each dimension of the occupation vector, which are dynamically adjusted as the algorithm progresses. $\theta_t$ tracks each dimension's occupation, attributing higher weights to the dimensions that were most used. Those weights $\theta_t$ are used to

calculate a "dimension-adjusted cost" $\langle \theta_t, a_t \rangle$ for each item, which goes into the final cost $Z\langle \theta_t, a_t \rangle$ of an item. $Z$ can be viewed as the estimated trade-off between item's rewards and occupations, and is set to a fixed scalar value. If an item's reward is higher than its final cost $Z\langle \theta_t, a_t \rangle$, the algorithm picks that item. Also, **there is a value for $Z$ that minimizes AD's regret, and it is $Z^* = \frac{\text{OPT}}{B}$**. However, since $Z^*$ depends on the knowledge of OPT, the algorithm tries to estimate OPT using the first $\delta T$ time steps.

Notice AD works in two phases: an observation phase and an active phase. During the observation phase, the algorithm doesn't pick any item. When the observation phase is done, the algorithm sets a value for $Z$, using an estimation $\text{OPT}'$ of OPT, based on the items seen so far. The estimation for OPT is obtained by solving a scaled version of problem to find $\text{OPT}'$ (Definition 3.5), and then estimating that $\text{OPT} \approx \frac{\text{OPT}'}{\delta}$. The scaled version of the problem, used to find $\text{OPT}'$, is presented bellow:

**Definition 3.5** ($\text{OPT}'$)**.** *We denote by* $\text{OPT}'$ *the value of the offline optimum solution to the following problem, which is related to the first $\delta T$ time steps:*

$$\begin{aligned} &\max \sum_{t=1}^{\delta T} c_t x_t \\ &\textit{s.t.} \sum_{t=1}^{\delta T} a_t x_t \leq \left(\delta B + \eta\sqrt{\delta B}\right) \mathbf{1}, \end{aligned}$$

*where* $\eta = \sqrt{3\log\left(\frac{d+2}{\varepsilon^2}\right)}$ *and* $\delta = 12\frac{4\varepsilon^2}{\log d}\log\left(\frac{d+2}{\varepsilon^2}\right)$.

The weights attributed to the occupation dimensions are managed using the MWU algorithm [3]. Since more occupied dimensions are attributed a higher weight, if an item has high occupation in a dimension already very occupied, its adjusted cost $\langle \theta, a_t \rangle$ gets higher, and the algorithm has an incentive to skip that item. Thus, the algorithm is encouraged to keep the occupation balanced between the available dimensions.

Considering that Algorithm AD was designed for the RANDOMORDER model, at first, we deem it unsuitable for the MIXED model. An adversary aware of the observation phase would choose to manipulate the first $\delta T$ time steps, making them adversarial, ruining the OPT estimation, resulting in a bad choice of $Z$ by the algorithm. Our hypothesis is that a bad choice of $Z$ would make AD perform poorly during the active phase. That is the main motivation behind our work, and we test that hypothesis later, with experiments, on Section 6.

# 4
# Algorithm AD$m$ for the MIXED Model

In this section we prove that even though AD was designed originally for the RANDOMORDER model, a variation AD$m$ of AD has good guarantees in the MIXED model. To obtain a good final guarantee, AD$m$ still requires a parameter $Z$ to be given beforehand such that $Z \approx Z^* = \frac{\text{OPT}_{Stoch}}{B}$, which implies the knowledge of $\text{OPT}_{Stoch}$. Since it is not easy to estimate $\text{OPT}_{Stoch}$ (because we do not know which times are random-order), this is a non-trivial issue that will be addressed in the next section.

Actually, this estimation of $\text{OPT}_{Stoch}$ will require us to run AD$m(Z)$ on multiple time intervals. More precisely, given an interval $I \subseteq [T]$ of the time steps, our aim is to approximately solve the Online Packing IP only considering the items in $I$ but using the appropriately scaled capacity $\frac{|I|}{T}B\mathbf{1}$. That is, we want to solve the following problem (with columns coming online):

$$\begin{aligned} \max & \sum_{t \in I} c_t x_t \\ \text{s.t.} & \sum_{t \in I} a_t x_t \leq \frac{|I|}{T} B\mathbf{1} \\ & x_t \in \{0,1\} \quad \forall t \in I. \end{aligned} \tag{IP(I)}$$

We use $\text{OPT}_{Stoch}(I)$ to denote the offline optimal solution that considers only the RANDOMORDER time steps of this problem. We use $\lambda_I$ to denote the fraction of adversarial time steps in interval $I$. **Again we use $r_t = c_t x_t$ (which is in $\{0, c_t\}$) to denote the reward obtained at time $t$ by a solution $x$, and $v_t = a_t x_t \in [0,1]^d$ to denote the contribution to the occupation incurred by this time step.**

AD$m(Z, I)$ is described as Algorithm 2, below. It is composed by only the active phase of AD, since it receives $Z$ as an argument. In addition, the maximum occupation it allows is in accordance with problem IP(I). Notice that AD$m$ uses a version of MWU adapted to gains in the range of $[-3, 3]$ and with $d + 1$ experts, in accordance with Lemma 3.2.

**Algorithm 2** Algorithm AD$m$ for an interval in the Mixed Model

**procedure** AD$m(B, T, Z, I)$
  Initialize $\theta_1$ from MWU with $d+1$ experts and gains in range $[-3, 3]$
  **for** $t \in I$ **do**
    Pack item if $c_t - Z\langle \theta_t, a_t \rangle \geq 0$
    **if** any dimension of the occupation exceeds $\frac{|I|B}{T}$ **then**
      Exit
    Compute the gains $h_{t,j} = v_{t,j} - \frac{B}{T}$ of each dimension $j = 1, \ldots, d$ and set $h_{t,0} = 0$
    Send $h_t$ to the MWU to obtain the next occupation weights $\theta_{t+1}$

The main result of this section is a high probability lower bound for AD$m$, that follows. Recall $\text{OPT}_{Stoch}$ is the optimal solution for the whole problem, with $T$ time steps (and not just over the interval $I$), considering only the RANDOMORDER time steps. Also, without loss of generality, we assume throughout this section that $T > B$, otherwise picking all the items is a feasible solution and obtaining the optimal result would be trivial.

**Theorem 4.1.** *Consider an instance* (IP(I)) *of the Online Packing IPs problem over an interval $I$ in the* MIXED *model. Assume $|I| \leq \frac{T}{2}$ and $c_t \in [0, \frac{\text{OPT}_{Stoch}}{B}]$. Let $\lambda \leq \frac{1}{2}$ denote the fraction of adversarial time steps for the whole instance, and $\lambda_I$ denote the fraction of adversarial time steps within interval $I$.*

*Then, for any $\varepsilon \in (0, \frac{1}{10}]$ and any $Z \geq 0$, if $B \geq \Omega(\frac{T}{|I|}\frac{\log d/\delta}{\varepsilon^2(1-\lambda)})$, algorithm* AD$m$ *always produces a feasible solution, and with probability at least $1-\delta$ it has value*

$$\text{AD}m(Z, I) \geq \min\left\{\frac{|I|\,\text{OPT}_{Stoch}}{T}\ ,\ (1-\varepsilon)\frac{|I|ZB}{T}\right\} - \frac{\lambda_I|I|}{T}\,\text{OPT}_{Stoch} - Z\frac{3\log(d+1)}{\varepsilon} - 2\lambda\frac{|I|}{T}ZB - O\left(\varepsilon\frac{|I|}{T}(\text{OPT}_{Stoch} + ZB)\right).$$

First, notice $Z$ is present in both positive and negative terms, so indeed there is a $Z^*$ maximizes the expression. Next, notice the bound is valid with probability $1-\delta$, and $B \geq \Omega(\frac{T}{|I|}\frac{\log d/\delta}{\varepsilon^2(1-\lambda)})$, meaning, if $B$ increases, the probability that AD$m$ respects the lower bound increases. In contrast, if we have more dimensions, the problem gets harder to solve, and our guarantee slowly deteriorates.

In addition, suppose we use Theorem 4.1 in a completely RANDOMORDER instance, and suppose we use $Z = \frac{\text{OPT}_{Stoch}}{B}$ and $B \geq \frac{T}{|I|}\frac{\log(d+1)}{\varepsilon^2(1-\lambda)}$. Theorem 4.1 becomes

$$\text{AD}m(Z, I) \geq (1 - \lambda_I - 2\lambda - O(\varepsilon))\frac{|I|}{T}\text{OPT}_{Stoch}.$$

On a completely RANDOMORDER instance, $\lambda = \lambda_I = 0$. Then, taking the sum over all $\frac{T}{|I|}$ intervals, we recover the result of the original AD algorithm, which is

$$\text{AD}(Z) \geq (1 - O(\varepsilon))\text{OPT}_{Stoch},$$

and note that $\text{OPT}_{Stoch} = \text{OPT}$ in that case, since every time step is stochastic.

For the remainder of this section we prove Theorem 4.1. Let $(r_t, v_t)$ be the rewards and occupation incurred by the algorithm at time $t$. Also, let us define $(r_t^*, v_t^*)$ as the option picked by $\text{OPT}_{Stoch}$ in each random-order time step $t \in Stoch$ (over the whole problem, not just the interval $I$). More formally:

$$(r_t^*, v_t^*) = \begin{cases} \text{option picked by OPT}_{Stoch} \text{ in time step } t\,, & \text{if } t \in Stoch \\ (0,0), & \text{if } t \in Adv \end{cases}$$

The starting point is the following lemma, which relates the reward obtained by AD$m$ to the following "Lagrangian value" $\mathcal{L}_I$ of the optimal solution restricted to the interval $I$, namely:

$$\mathcal{L}_I := \sum_{t \in I, t \leq \tau} \left( r_t^* - Z\langle \theta_t, v_t^* - \tfrac{B\mathbf{1}}{T} \rangle \right).$$

**Lemma 4.2.** *Consider the same assumptions as in Theorem 4.1. Assume we are running* AD$m$ *for an interval $I$. Let $\tau_I$ be the stopping time of* AD$m$ *relative to interval $I$, and let $[\tau_I]$ be the sequence of time steps in $I$ up to $\tau_I$. Then in every scenario, the reward obtained by* AD$m$ *is at least*

$$\sum_{t \in [\tau_I]} r_t \geq \mathcal{L}_I + (1 - \varepsilon) Z B \left( \frac{|I| - \tau_I}{T} \right) - Z \frac{3 \log(d+1)}{\varepsilon}.$$

*Proof.* Recall the decision criteria for AD$m$ (Algorithm 2) to choose an item is to pack it only if $c_t - Z\langle \theta_t, a_t - \frac{B\mathbf{1}}{T} \rangle \geq 0$. In other words, the algorithm assigns to $(r_t, v_t)$ either $(c_t, a_t)$ or $(0,0)$ in time $t$, and it chooses the option that maximizes the expression $r_t - Z\langle \theta_t, v_t - \frac{B\mathbf{1}}{T} \rangle$, like so:

$$(r_t, v_t) = \text{argmax}_{(r,v) \in \{(c_t, a_t), (0,0)\}} \, r - Z\langle \theta_t, v - \tfrac{B\mathbf{1}}{T} \rangle$$

Since we are maximizing the above expression, we can state that, for every time step $t$,

$$r_t - Z\langle \theta_t, v_t - \tfrac{B\mathbf{1}}{T} \rangle \geq r_t^* - Z\langle \theta_t, v_t^* - \tfrac{B\mathbf{1}}{T} \rangle,$$

and so adding over all times in $I$ up to $\tau_I$ and reorganizing the terms gives

$$\sum_{t\in[\tau_I]} r_t \geq \underbrace{\sum_{t\in[\tau_I]} \left( r_t^* - Z\langle \theta_t, v_t^* - \tfrac{B\mathbf{1}}{T}\rangle \right)}_{\mathcal{L}_I} + \sum_{t\in[\tau_I]} Z\langle \theta_t, v_t - \tfrac{B\mathbf{1}}{T}\rangle. \tag{4-1}$$

We claim that the second summation on the right-hand side can be lower bounded as

$$\sum_{t\in[\tau_I]} \langle \theta_t, v_t - \tfrac{B\mathbf{1}}{T}\rangle \geq (1-\varepsilon)\left(\frac{|I|B}{T} - \frac{\tau_I B}{T}\right) - \frac{3\log(d+1)}{\varepsilon}. \tag{4-2}$$

To see this, recall that $h_t = v_t - \frac{B}{T}\mathbf{1}$, therefore the left-hand side in the required inequality is $\sum_{t\in[\tau_I]}\langle h_t, \theta_t\rangle$. Notice that for every $\theta \in \blacktriangle^d$, the absolute value of $\langle h_t, \theta\rangle$ is at most $\langle h_t, \theta\rangle + \frac{2B}{T}$, because when $v_t = \{0\}^d$, $|\langle h_t, \theta\rangle| = \frac{B}{T} \leq \langle v_t, \theta\rangle - \langle \frac{B}{T}, \theta\rangle + \frac{2B}{T} = \langle h_t, \theta\rangle + \frac{2B}{T}$. Using the hypotheses $v_t \in [0,1]^d$ and $B \leq T$, we arrive at $|\langle h_t, \theta\rangle| \leq 1 + \frac{2B}{T} \leq 3$. Now if we let $\theta^* := \operatorname{argmax}_{\theta\in\blacktriangle^d} \sum_{t\in[\tau_I]}\langle h_t, \theta\rangle$, we can use the guarantee from Lemma 3.2 to get

$$\sum_{t\in[\tau_I]} \langle h_t, \theta_t\rangle \geq (1-\varepsilon)\sum_{t\in[\tau_I]} \langle h_t, \theta^*\rangle - \frac{3\log(d+1)}{\varepsilon}. \tag{4-3}$$

Moreover, when the algorithm stops, there are two possible, but mutually exclusive, situations:

1. The algorithm stopped early due to a constraint violation. Let $e_j$ to be the $d$-dimensional vector with all entries 0, except for the $j$-th entry, which is 1. If some occupation dimension $j$ violated its constraint, then $\sum_{t\in[\tau_I]}\langle v_t, e_j\rangle \geq \frac{|I|B}{T}$ at the stopping time $\tau_I$. Now, looking at the MWU gain functions, $\sum_{t\in[\tau_I]}\langle h_t, \theta^*\rangle \geq \sum_{t\in[\tau_I]}\langle h_t, e_j\rangle \geq \sum_{t\in[\tau_I]}\langle v_t, e_j\rangle - \sum_{t\in[\tau_I]}\langle \frac{B}{T}, e_j\rangle \geq \frac{|I|B}{T} - \frac{\tau_I B}{T}$.

2. The algorithm stopped at the end of the interval $I$, that is, $\tau_I = |I|$, in which case $\sum_{t\in[\tau_I]} v_t \leq \frac{|I|B\mathbf{1}}{T}$. Because of that, $\sum_{t\in[\tau_I]}(v_t - \frac{B\mathbf{1}}{T}) \leq \mathbf{0}$, so the $\theta^*$ that maximizes $\sum_{t\in[\tau_I]}\langle h_t, \theta\rangle$ is $\theta^* = 0$, and hence $\sum_{t\in[\tau_I]}\langle h_t, \theta^*\rangle = 0 = \frac{|I|B}{T} - \frac{\tau_I B}{T}$ (recall that in this current case we have $\tau_I = |I|$).

Therefore, for both cases, we can state that $\sum_{t\in[\tau_I]}\langle h_t, \theta^*\rangle \geq \frac{|I|B}{T} - \frac{\tau_I B}{T}$. Using this bound on (4-3) then proves (4-2). Employing the latter on (4-1) concludes the proof of the lemma. ■

Next, we lower bound the "Lagrangian value" $\mathcal{L}_I$ of the optimal solution $(r_t^*, v_t^*)$. Recall that $\lambda$ is the fraction of adversarial time steps in the whole instance (i.e., there are $\lambda T$ adversarial time steps in the whole instance).

**Lemma 4.3.** *Let $\tau_I$ be the stopping time of* AD$m$ *relative to interval $I$. Let $\boldsymbol{s}_I$ be the number of stochastic time steps in $I$ up to (and possibly including) the time $\tau_I$. Assume $c_t \in [0, \frac{\text{OPT}_{Stoch}}{B}]$, $\varepsilon \in (0, \frac{1}{10}]$, $B \geq \Omega(\frac{T}{|I|}\frac{\log d/\delta}{\varepsilon^2(1-\lambda)})$, $|I| \leq (1-\lambda)\frac{T}{2}$. Then, with probability at least $1 - \frac{\delta}{2}$, we have*

$$\mathcal{L}_I \geq \boldsymbol{s}_I \frac{\text{OPT}_{Stoch}}{T} - 2\lambda\frac{|I|}{T}ZB - O\Big(\varepsilon\tfrac{|I|}{T}(\text{OPT}_{Stoch} + ZB)\Big).$$

The proof of this lemma is quite technical and deferred to Appendix B. Here, instead, we give an informal idea of the proof to convey the main ideas.

*Idea of the proof.* By definition, we have that $\sum_{t\in Stoch} r_t^* = \text{OPT}_{Stoch}$. Since the items in the random-order part of the instance are sampled (without replacement), we see that the expected reward $\mathbb{E}r_t^*$ is the same for all the $(1-\lambda)T$ random-order times $t$. Then, together with the previous equality this implies that $\mathbb{E}r_t^* = \frac{\text{OPT}_{Stoch}}{(1-\lambda)T}$ for all random-order times $t$. For adversarial times $t$, by definition $r_t^* = 0$.

Similarly, the optimal solution $(r_t^*, v_t^*)$ certainly respects the constraints of the problem, meaning that $\sum_{t\in Stoch} v_t^* \leq B\mathbf{1}$. Again, since the items in the random-order part of the instance are sampled (without replacement), the expected vector $\mathbb{E}v_t^*$ is the same for all the $(1-\lambda)T$ random-order times $t$. Thus, the previous inequality implies that $\mathbb{E}v_t^* \leq \frac{B\mathbf{1}}{(1-\lambda)T}$. For adversarial times $t$, by definition $v_t^* = 0$, so in particular $\mathbb{E}v_t^* = 0$. Thus, we can state that $\mathbb{E}v_t^* \leq \frac{B}{(1-\lambda)T}$ for *any* time step $t$.

Putting these two observations together gives that the expected Lagrangian value $\mathcal{L}_I$ is, roughly speaking, at least

$$\begin{aligned}
\mathbb{E}\mathcal{L}_I &= \mathbb{E}\sum_{t\leq\tau_I, t\in Stoch} r_t^* - \mathbb{E}\sum_{t\leq\tau_I} Z\langle\theta_t, v_t^* - \tfrac{B\mathbf{1}}{T}\rangle \\
&\gtrsim \boldsymbol{s}_I \cdot \mathbb{E}r_t^* - \mathbb{E}\sum_{t\leq\tau_I} Z\langle\theta_t, v_t^* - \tfrac{B\mathbf{1}}{T}\rangle \\
&\gtrsim \boldsymbol{s}_I \cdot \frac{\text{OPT}_{Stoch}}{T} - \sum_{t\leq\tau_I} Z\langle\mathbb{E}\theta_t, \underbrace{\mathbb{E}v_t^* - \tfrac{B\mathbf{1}}{T}}_{\lesssim 0}\rangle \\
&\gtrsim \boldsymbol{s}_I \cdot \frac{\text{OPT}_{Stoch}}{T},
\end{aligned}$$

where we need to justify several things. In the first inequality, because of the random stopping time $\tau_I$, it is not clear that we can "move the expectation inside". Even more problematic, is the second inequality, where to move the expectation inside the inner product we further assumed that the random variables $\theta_t$ and $v_t^*$ were independent on a random-order time $t$; however this is definitely not the case: since items are sampled without replacement, information about items up to time $t-1$ (which determine $\theta_t$) offers some information about which item will appear at time $t$, and hence some information about $v_t^*$.

Nonetheless, ignoring these issues, this essentially prove the lemma in expectation. In the full proof we make these steps formal, and also prove the lemma with high probability instead of in expectation. ■

Now we are ready to prove Theorem 4.1.

***Proof of Theorem 4.1***. Let $[\tau_I]$ denote the sequence of time steps in $I$ up to, and possibly including, $\tau_I$. Recall $\tau_I$ is the stopping time of algorithm AD running in interval $I$. Substituting the inequality from Lemma 4.3 in Lemma 4.2, we get that with probability at least $1 - \frac{\delta}{2}$

$$\sum_{t\in[\tau_I]} r_t \geq \boldsymbol{s}_I \frac{\text{OPT}_{Stoch}}{T} + (1-\varepsilon)ZB\left(\frac{|I|-\tau_I}{T}\right) - Z\frac{3\log(d+1)}{\varepsilon} - 2\lambda\frac{|I|}{T}ZB - O\left(\varepsilon\frac{|I|}{T}(\text{OPT}_{Stoch}+ZB)\right).$$

We know that $\boldsymbol{s}_I \geq \tau_I - \lambda_I|I|$. Substituting in the above, we get:

$$\begin{aligned}\sum_{t\in[\tau_I]} r_t &\geq (\tau_I - \lambda_I|I|)\frac{\text{OPT}_{Stoch}}{T} + (1-\varepsilon)ZB\left(\frac{|I|-\tau_I}{T}\right) - Z\frac{3\log(d+1)}{\varepsilon} \\ &\quad - 2\lambda\frac{|I|}{T}ZB - O\left(\varepsilon\frac{|I|}{T}(\text{OPT}_{Stoch}+ZB)\right) \\ &= \frac{\tau_I}{|I|}\frac{|I|\,\text{OPT}_{Stoch}}{T} + (1-\varepsilon)\frac{|I|ZB}{T}\left(1-\frac{\tau_I}{|I|}\right) - \frac{\lambda_I|I|}{T}\text{OPT}_{Stoch} - Z\frac{3\log(d+1)}{\varepsilon} \\ &\quad - 2\lambda\frac{|I|}{T}ZB - O\left(\varepsilon\frac{|I|}{T}(\text{OPT}_{Stoch}+ZB)\right)\end{aligned}$$

Notice that the first two terms have the form $x\cdot a + b\cdot(1-x)$ for some $x \in [0,1]$. This expression is always greater than $\min\{a,b\}$, since it is a weighted average between $a$ and $b$. Using that, we arrive at the statement of the theorem:

$$\sum_{t\in[\tau_I]} r_t \geq \min\left\{\frac{|I|\,\text{OPT}_{Stoch}}{T}\ ,\ (1-\varepsilon)\frac{|I|ZB}{T}\right\} - \frac{\lambda_I|I|}{T}\text{OPT}_{Stoch} - Z\frac{3\log(d+1)}{\varepsilon} - 2\lambda\frac{|I|}{T}ZB - O\left(\varepsilon\frac{|I|}{T}(\text{OPT}_{Stoch}+ZB)\right).$$

■

# 5
# Proposed Robust Algorithm for the MIXED Model

In this section we present the proposed algorithm for the MIXED model and prove a high probability lower bound for its rewards. We make use of AD$m$ and solve the problem of providing AD$m$ with a "good enough" $Z$ value, which depends on $\text{OPT}_{Stoch}$.

Let $Z^*$ be the optimal fixed value for $Z$ in hindsight. The idea of the proposed algorithm is to start AD$m$ with a random $Z$ and learn $Z^*$ in an online fashion. To learn $Z^*$, we consider a discrete set $\mathcal{Z}$ of possible values, under the framework of *integral prediction with expert advice* (Section 3.2). Then, we use the MSMW algorithm (Lemma 3.4) to learn the best approximation of $Z^*$ that is present in $\mathcal{Z}$. We denote this value by $\tilde{z}$. After presenting the algorithm formally, we prove the following result:

**Theorem 5.1** (Main Theorem)**.** *Assume* $\varepsilon \in (0, \frac{1}{10})$ *and assume the fraction of adversarial times* $\lambda \leq \frac{1}{2}$*. Suppose that* $c_t \in [0, \frac{\text{OPT}_{Stoch}}{B}]$ *and* $B \geq \Omega(\frac{(\log\log T + \log(1/\varepsilon))\log d}{\varepsilon^4})$*. Also suppose we have a* $\text{poly}(n)$ *estimate of* $\text{OPT}_{Stoch}$*, denoted by* $\widehat{\text{OPT}}_{Stoch}$*, such that* $\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}} \leq \text{OPT}_{Stoch} \leq T^{10}\widehat{\text{OPT}}_{Stoch}$*.*

*Then, algorithm* ROBUSTAD *produces a feasible solution, and if* $B \geq \Omega(\frac{1}{\varepsilon^2}\log[\frac{(\log\log T - \log\varepsilon)}{\varepsilon^2}\frac{d}{\delta'}])$*, then, with probability at least* $1-\delta'$*, its value is*

$$\text{ROBUSTAD} \geq (1 - 5\lambda - O(\varepsilon))\text{OPT}_{Stoch}$$



## 5.1
## Algorithm

First, let us group the time steps into $K$ intervals, such that we are left with $K$ instances of the problem (IP(I)) (for simplicity we assume $K$ divides $T$). Recall the $i$-th interval will have its own $\lambda_i$ (fraction of adversarial times). Our algorithm, then, at a high-level consists of the following steps:

1. Run AD$m(Z)$ for one whole interval.
2. Choose the next $Z$ value through MSMW.
3. Repeat from Step (1) but using the new value of $Z$.

**Discretization of experts.** In order to learn a good $Z^*$ approximation using a *prediction with expert advice* perspective, we need a discrete set of experts. We propose a discretization of some range of $\mathbb{R}_+$, such that each discretized value is a candidate for the $Z^*$ approximation. Moreover, each discretized value will be assigned to an expert, that predicts its associated value repeatedly.

Let $\mathcal{Z}$ denote a discrete set of candidate values to approximate $Z^*$. Since $Z^*$ may or may not be contained in $\mathcal{Z}$, we aim to learn $\tilde{z} = \min\{z \in \mathcal{Z} | z \geq Z^*\}$. Recall that in hindsight the "right" $Z^*$ is $\frac{\text{OPT}_{Stoch}}{B}$.

Additionally, when thinking about how numerous or granular should $\mathcal{Z}$ be, we are faced with a trade-off: making the size of $\mathcal{Z}$ small lowers MSMW's regret, but increases the discretization error (due to the greater distance from $\tilde{z}$ to $Z^*$). On the other hand, making $\mathcal{Z}$ more numerous reduces the discretization error, but increases MSMW's regret.

In order to choose the range of $\mathbb{R}^+$ to be discretized by the experts, we assume knowledge of $\widehat{\text{OPT}}_{Stoch}$, a $\text{poly}(n)$ estimate of $\text{OPT}_{Stoch}$, such that

$$\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}} \leq \text{OPT}_{Stoch} \leq T^{10}\widehat{\text{OPT}}_{Stoch}.$$

Assumptions about having an estimate of the optimum available are also present in other works concerning Online Packing IPs, and [14] provides a detailed discussion and some techniques to work around it. We reproduce one of their arguments stating that, in most applications, previous data is available, allowing such an estimate to be made. Finally, we choose

$$\mathcal{Z} = \{\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}}, 2\,\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}}, 2^2\,\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}}, \ldots, T^{10}\,\widehat{\text{OPT}}_{Stoch}\},$$

namely discretizing using powers of 2. We collect the following properties of this set that will be useful later.

**Lemma 5.2.** *The discretization set $\mathcal{Z}$ has size $\log_2 T^{20} = \Theta(\log T)$. Moreover, if $\tilde{z} = \min\{z \in \mathcal{Z} | z \geq Z^*\}$, it offers the following approximation for $Z^* = \frac{\text{OPT}_{Stoch}}{B}$: there is $\tilde{z} \in \mathcal{Z}$ such that*

1. $\tilde{z} \geq \frac{\text{OPT}_{Stoch}}{B}$
2. $\tilde{z} \leq \frac{2\,\text{OPT}_{Stoch}}{B}$

*Proof.* To find the size of the set $\mathcal{Z}$ is to find $|\mathcal{Z}|$ such that $\frac{\widehat{\text{OPT}}_{Stoch}}{T^{10}}\ 2^{|\mathcal{Z}|} = T^{10}\ \widehat{\text{OPT}}_{Stoch}$, so

$$|\mathcal{Z}| = \log_2 T^{20} = 20(\frac{\log T}{log2}),$$

which is $\Theta(\log T)$.

Also, let $\ddot{z}$ and $\tilde{z}$ be the closest values to $Z^*$ in the set $\mathcal{Z}$, such that $\ddot{z} \leq Z^* \leq \tilde{z}$, with $\tilde{z}$ possibly being equal to $Z^*$. By definition, $\tilde{z} \geq Z^*$, and $\ddot{z} \leq Z^*$. By the way the set was constructed, we know that $\ddot{z} = \frac{\tilde{z}}{2}$, resulting in $\tilde{z} = 2\ \ddot{z} \leq 2\ Z^*$. Lastly, using $Z^* = \frac{\text{OPT}_{Stoch}}{B}$ proves both bounds for $\tilde{z}$. ■

**Gains Function.** We need a gains function to evaluate how well each expert did in a given interval and to feed that evaluation to MSMW. The gains function should map a $Z$ value and an interval to a score representing the performance of that $Z$ value in the given interval. Let $\text{AD}m(Z, I_i)$ denote the reward obtained by algorithm $\text{AD}m(Z)$ with parameter $Z$ applied to the $i$-th interval. The gains function we chose is a truncation of $\text{AD}m(Z)$, denoted by $\overline{\text{AD}m(Z)}$:

$$\overline{\text{AD}m(Z, I_i)} = \min\left(\text{AD}m(Z, I_i)\ , \frac{|I|B}{T}Z\right)$$

Note that different $Z$ values yield different truncation thresholds and also result in different items being picked by $\text{AD}m$, which consequently results in different gains. Also note that in order to compute the gains of each expert for the $i$-th interval, the algorithm has to run a simulation of $\text{AD}m(Z, I_i)$ for each $Z$ candidate. We call this a simulation because the items picked during those runs do not account for the final reward or occupation of the algorithm. They merely answer the question of "What reward would we gain if we used this other $Z$ expert instead, on the $i$-th interval?". And surely, to be able to run this simulation, the items of the interval need to be known, so this simulation will run for every $Z$ candidate when the algorithm reaches the end of an interval, and that interval. We present the algorithm below.

**Algorithm 3** Robust Algorithm for the Mixed Model

1: **procedure** ROBUSTAD($B$,$T$,$\widehat{\text{OPT}}_{Stoch}$)
2: Using $\widehat{\text{OPT}}_{Stoch}$ create the discretized set of experts $\mathcal{Z}$
3: Call the procedure MSMW over $\mathcal{Z}$ to obtain the expert $Z_1$ for the first round
4: Set the number of intervals $K \leftarrow \frac{\log\log T + \log(1/\varepsilon)}{\varepsilon^2}$
5: **for all** $i = 1, 2, \ldots, K$ **do** ▷ Consider each interval
6: Run algorithm $\text{AD}m(Z_i)$ over the current interval $I_i$, and pick the items according to its decisions
7: At the end of the interval, compute the rewards for all the experts $Z \in \mathcal{Z}$. That is, compute $(h_i)_Z := \overline{\text{AD}m(Z, I_i)}$ for all $Z \in \mathcal{Z}$
8: Send the reward vector $h_i$ to the algorithm MSMW to obtain the next expert $Z_{i+1}$

## 5.2 Theoretical Guarantees: Proof of Theorem 5.1

The outline of the main proof is:

1. Since we run $\text{AD}m(Z)$ on each interval, our total reward is $\text{ROBUSTAD} = \sum_{i=1}^{K} \text{AD}m(Z_i, I_i) \geq \sum_{i=1}^{K} \overline{\text{AD}m(Z_i, I_i)}$, so it suffices to lower bound the last term.

2. We use the MSMW guarantee with gains function $(h_i)_Z = \overline{\text{AD}m(Z, I_i)}$ to obtain

$$\sum_{i=1}^{K} \overline{\text{AD}m(Z_i, I_i)} \geq (1-\varepsilon) \sum_{i=1}^{K} \overline{\text{AD}m(\tilde{z}, I_i)} - O\left(\frac{c_{\tilde{z}} \log(n/\varepsilon)}{\varepsilon}\right),$$

   where $c_{\tilde{z}}$ is an upper bound for all $(h_i)\tilde{z}$ of the intervals. Recall $\tilde{z} \in \mathcal{Z}$ is good approximation of $Z^*$ present in $\mathcal{Z}$.

3. Lower bound the term $\sum_i^K \overline{\text{AD}m(\tilde{z}, I_i)}$ in the right-hand side using Theorem 4.1 with $Z = \tilde{z}$, and upper bound the last term (more precisely $c_{\tilde{z}}$) using the the truncation imposed.

We now present a complete proof of the theorem.

*Proof of Theorem 5.1.* First, by the definition of ROBUSTAD, we know total reward it obtains is

$$\text{ROBUSTAD} = \sum_{i=1}^{K} \text{AD}m(Z_i, I_i) \geq \sum_{i=1}^{K} \overline{\text{AD}m(Z_i, I_i)}. \tag{5-1}$$

Now let $\tilde{z}$ be the expert in the discretized set $\mathcal{Z}$ described by Lemma 5.2. Because of the truncation applied to the gains function that evaluates $Z$ candidates, the largest reward expert $\tilde{z}$ gets is

$$\max_{i \leq K} \overline{\text{AD}m(\tilde{z}, I_i)} \leq \frac{|I|B}{T}\tilde{z}.$$

Therefore, using the MSMW guarantee from Lemma 3.4, we get

$$\sum_{i=1}^{K} \overline{\text{AD}m(Z_i, I_i)} \geq (1-\varepsilon) \sum_{i=1}^{K} \overline{\text{AD}m(\tilde{z}, I_i)} - \frac{|I|B}{T}\tilde{z} \cdot O\left(\frac{\log(|\mathcal{Z}|/\varepsilon)}{\varepsilon}\right)$$

in which we can use that $\tilde{z} \leq \frac{2\text{OPT}_{Stoch}}{B}$ and $|\mathcal{Z}| = O(\log T)$, from Lemma 3.4, to get

$$\sum_{i=1}^{K} \overline{\text{AD}m(Z_i, I_i)} \geq (1-\varepsilon) \sum_{i=1}^{K} \overline{\text{AD}m(\tilde{z}, I_i)} - \text{OPT}_{Stoch} \cdot O\left(\frac{\log\log T + \log(1/\varepsilon)}{\varepsilon K}\right).$$

From the fact that $K \geq \Omega(\frac{\log\log T + \log(1/\varepsilon)}{\varepsilon^2})$ it follows that

$$\begin{aligned}
&\geq (1-\varepsilon)\sum_{i=1}^{K} \overline{\mathrm{AD}m(\tilde{z}, I_i)} - \mathrm{OPT}_{Stoch} \cdot O(\varepsilon) \\
&\geq (1-\varepsilon)\sum_{i=1}^{K} \min\left\{\mathrm{AD}m(\tilde{z}, I_i), \frac{\mathrm{OPT}_{Stoch}}{K}\right\} - \mathrm{OPT}_{Stoch} \cdot O(\varepsilon),
\end{aligned} \tag{5-2}$$

where the last inequality follows from the truncation in the definition of $\overline{\mathrm{AD}m(\cdot)}$ and the guarantee $\tilde{z} \geq \frac{\mathrm{OPT}_{Stoch}}{B}$. Applying Theorem 4.1 to interval $I_i$ with $Z = \tilde{z}$ and $\delta = \frac{\delta'}{K}$, we have that with probability at least $1 - \frac{\delta'}{K}$ we have

$$\begin{aligned}
\mathrm{AD}m(\tilde{z}, I_i) &\geq \mathrm{OPT}_{Stoch}\left[\frac{(1-\varepsilon)}{K} - \frac{\lambda_{I_i}|I_i|}{T} - \frac{2\log(d+1)}{\varepsilon B} - \frac{4\lambda}{K} - O\left(\frac{\varepsilon}{K}\right)\right] \\
&\geq \frac{\mathrm{OPT}_{Stoch}}{K} - \mathrm{OPT}_{Stoch}\left[\frac{\lambda_{I_i}}{K} + \frac{4\lambda}{K} + O\left(\frac{\varepsilon}{K}\right)\right],
\end{aligned}$$

where the last inequality uses the assumption $B \geq \Omega(\frac{K\log d}{\varepsilon^2})$. Note that indeed we can apply the theorem with $\delta = \frac{\delta'}{K}$ because we assumed here that $B \geq \Omega(\frac{\log(Kd/\delta')}{\varepsilon^2})$. Taking a union bound over all the $K$ intervals $I_1, \ldots, I_K$, we see that with probability at least $1 - \delta'$ this bound holds for all these intervals. In this case, employing this bound together with (5-1) and (5-2) we get

$$\begin{aligned}
\textsc{robustad} &\geq (1 - O(\varepsilon))\mathrm{OPT}_{Stoch} - \mathrm{OPT}_{Stoch} \cdot \sum_{i=1}^{K}\left[\frac{\lambda_{I_i}}{K} + \frac{4\lambda}{K} + O\left(\frac{\varepsilon}{K}\right)\right] \\
&= (1 - O(\varepsilon))\mathrm{OPT}_{Stoch} - \mathrm{OPT}_{Stoch} \cdot \left[5\lambda + O(\varepsilon)\right] \\
&= \left(1 - 5\lambda - O(\varepsilon)\right)\mathrm{OPT}_{Stoch}
\end{aligned}$$

with probability $1 - \delta'$. This concludes the proof of Theorem 5.1. ■

# 6
# Experiments

Throughout this section, we present results of computational experiments comparing ROBUSTAD with AD. We detail experiments on various types of instances, as we investigate the robustness, advantages and drawbacks of each algorithm. First, we introduce some practical improvements on ROBUSTAD, and we run sensitivity tests using Wang [27] instances. Then, we use those results to set the parameters $\varepsilon$ for AD and ROBUSTAD, and we also set $K$ for ROBUSTAD. Afterwards, we present experiments with Chu and Beasley [10] instances that were manipulated, in order to simulate some MIXED model scenarios.

The environment in which experiments were conducted was an Ubuntu 18 running inside a Windows 10 through WSL 2 (Windows Subsystem for Linux). The machine has an Intel i5-7400 processor (4 cores at 3.00 GHz each) and 24Gb of RAM. Due to the high number of decision variables of the instances (i.e. $T = 10000$), solving the instances as integer programs would not allow us to run all the experiments in a viable time. Therefore, all the optimums presented in this section were obtained by solving the instances as linear programs instead. The solver used was Coin-OR's Cbc [1], integrated into the Python-MIP package [2]. The language chosen to generate the instances and run the algorithms in was Python 3.9.

The metric we use to evaluate the algorithms is the Relative Loss. For an algorithm $\mathsf{Alg}(instance)$ and a set of instances $S$, the Relative Loss can be defined as:

$$RL_S = 1 - \frac{1}{|S|} \sum_{i \in S} \frac{\mathsf{Alg}(i)}{\mathrm{OPT}_i},$$

that is, one minus the average score across all instances in $S$. Again, we emphasize that OPT will be approximated by the optimum of the linear program associated with the instance.

## 6.1
## Wang Instances

In this section, we run sensitivity tests for AD and ROBUSTAD's parameters, using instances proposed by Wang [27]. In addition, we use those tests to set the parameters for the next section. The instances proposed by Wang were designed to

[1] https://github.com/coin-or/Cbc
[2] https://www.python-mip.com/

fit the customer bidding model, which is simply another way of viewing the online packing problem.

The customer bidding model considers that there is a store with an inventory of $d$ products and $B_j$ of each product $j$ in stock. There is a line of $T$ customers and when a customer $t$ arrives at the counter, he states the products he wants ($a_t$) and bids a total price he is willing to pay ($c_t$). The store owner, then, decides to serve ($x_t = 1$) or not to serve ($x_t = 0$) that customer, and then proceeds to hear the next customer's proposal. The objective is for the store owner to make decisions trying to capture the maximum revenue, while being constrained to the availability of products in stock. Notice how this model fits perfectly into the Online Packing Problem, where items now are represented by customers.

The method used to generate those instances is the same used by Wang [27], except for a final extra step to ensure rewards are between $0$ and $1$. The full procedure is described below:

- (Step 1) Receive $T$, $d$, and $B$ as parameters.
- (Step 2) Set customer requests (item occupation): each entry of the cost matrix $A$ has value $1$ with probability $0.5$, and value $0$ otherwise. Each entry is set independently.
- (Step 3) Set the perceived prices of products: for each dimension $j$, generate an underlying price $p_j^* \leftarrow U(0, 1)$.
- (Step 4) Set customer offers (item rewards): $c_t = \langle p^*, a_t \rangle + N(0, 1)$, where $N(0, 1)$ is the standard normal distribution.
- (Step 5) Shift and normalize the rewards, to ensure they are in the range $[0, 1]$. It was done by applying the following transformation to each reward: $new\_c_t = \frac{c_t - c_{\min}}{c_{\max} - c_{\min}}$.

### 6.1.1 Tracking the Source of ROBUSTAD's Loss

Preliminary results turned out unsatisfactory regarding ROBUSTAD's performance. Setting the best $\varepsilon$ and $K$ (number of intervals) according to theory yielded relative losses from $40\%$ to $60\%$, on Wang instances. Given those results, before any parameter tuning, we try to identify the causes for the loss experienced by ROBUSTAD. In this section, all results will be performed with 10 permutations of a base Wang instance with $T = 10000$, $B = 1000$, and $d = 5$. Also, we will fix $\varepsilon = 0.25$ and $K = 150$ arbitrarily. At each step of the process, we make new modifications, measuring the performance changes relative to the modification. This way, we should be able to identify where the majority of losses comes from. Before making any modifications, we have a relative loss of $46.8\%$.

**Loss relative to learning $Z$.** On this first variation of the algorithm, we run ROBUSTAD with fixed $Z^* = \text{OPT}/B$ on every interval. This way, the difference to be observed in the performance will be relative to the learning of $Z$ only. We observed a relative loss of $29.7\%$. This $17.1$ percentage points of advantage suggests that maybe the algorithm is not learning $Z$ fast enough. While varying the number of intervals could be a valid attempt here, we deffer that experiment to the sensitivity tests in the next section, and here we try to increase the learning rate for the $Z$ learning mechanism. This rate was previously set equal to $\varepsilon$, at $0.25$. Our tests showed that, for running the algorithm without $Z^*$, using a value of 0.75 for the learning rate of $Z$ showed promising results, improving the relative loss from $46.8\%$ to $36.4\%$. We incorporate this learning rate for $Z$ into the algorithm.

**Remembering $\theta$ from previous intervals.** Next, we modify ROBUSTAD so that the learned weights attributed to the $\theta$ dimensions are not reset at the beginning of every interval. On one hand, we expect that with more time to learn $\theta^*$, performance could be better. On the other hand, changes in $Z$ could destabilize the learned weights for $\theta$, in the sense that weights learned for a previous value of $Z$ wouldn't be optimal for the new value of $Z$. With that modification, we observed no significant change in the relative loss. We conclude that resetting the $\theta$ learning is not one of the main sources of loss, thus, rejecting this modification.

**Loss due to splitting distributing the capacity to multiple intervals.** When we break $[T]$ into $K$ intervals and attribute a capacity of $B/K$ for each interval, we remove a huge degree of flexibility from the algorithm. That happens because unoccupied capacities from past intervals are lost. We can mitigate this effect by, at the end of every interval, reassigning the capacities of future intervals based on the current remaining capacity. So instead of every interval receiving a fixed capacity, each interval will receive its capacity according to the following:

- Let $m$ be the total occupation of the most occupied dimension until the current time step.
- At the beginning of the $i$-th interval, assign to it a capacity of $(B - m)/(K - i + 1)$.

Note that we simply allowed the interval to pack more items, but we did not change any gain functions of the MWU. With that modification, unused capacity of previous intervals are redistributed to the remaining intervals. Also note that the first interval receives a capacity of $B/K$, and following intervals receive at least $B/K$ of capacity. With this modification, the relative loss improved further more,

from $36.4\%$ to $22.9\%$. Therefore, this modification was accepted and incorporated into ROBUSTAD.

**Summary of modifications.** Altogether, two modifications applied to ROBUSTAD. The first was to increase the learning rate for $Z$ to $0.75$. The second was to redistribute unused capacities from previous intervals. When used together, those modifications were responsible for an improvement in relative loss, from $46.8\%$ to $22.9\%$.

### 6.1.2 Sensitivity Tests

For the sensitivity tests below, we use base instances with $T = 10000$, $B = 1000$ and $d = 5$ as default values for those settings. Additionally, our tests and methods are similar to the ones described in Wang [27], such that a comparison between both works is possible, but the methodology has little variations. Such variations emerge from the running speed of ROBUSTAD, that does not allow us to test against the same number of instances that Wang [27] uses. Next, we describe those differences in detail.

For the $\varepsilon$ and $K$ sensitivity tests, we generate one base instance, and then we shuffle the arrival order of that base instance to generate 10 new instances. For each value of $\varepsilon$ and $K$, the presented result is the average across those 10 instances. Wang [27] presents this test based on one instance also, but runs his algorithms on 500 permutations of that instance, for each $\varepsilon$.

For all the other sensitivity tests, we vary one setting of the base instances. For each different variation of the tested parameter, we generate 5 new instances, and then we shuffle each of those instances 5 times, generating a total of 25 instances. The final result presented is the average across those 25 instances. For those tests, Wang [27] generates 20 base instances, and shuffles each instance 20 times, reaching a total of 400 instances for each variation of the tested parameter.

Lastly, the $\varepsilon$ and $K$ sensitivity test will be used as a calibration step for ROBUSTAD, such that all the other tests will be done using a fixed $\varepsilon$ and $K$, chosen based on the results from their sensitivity test. In addition, we set AD's $\delta$ to $0.05$, meaning AD will use $5\%$ of each instance's time steps to estimate $Z$, while not picking any item. That value was chosen based on preliminary tests.

**$\varepsilon$ and $K$ sensitivity.** The first experiment presented will evaluate ROBUSTAD sensitivity to a change in $\varepsilon$ (the learning rate) and $K$ (the number of intervals). Notice that even though $K$ is set according to $\varepsilon$ in the algorithm's description, theoretically, $K$ is allowed to vary by a constant factor, which prompts the necessity

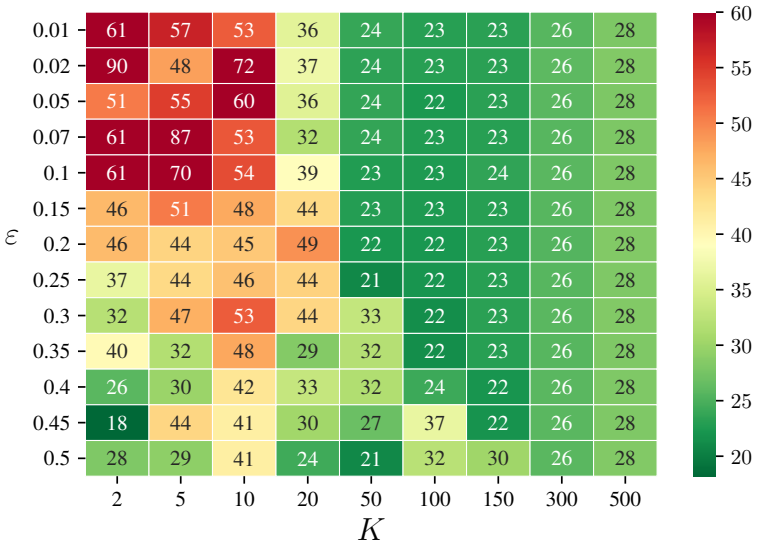


Figure 6.1: ROBUSTAD's relative loss (%) according to $\varepsilon$ and $K$.

to test $K$ separately from $\varepsilon$. Also, since we expect $K$ to be somewhat dependent on $\varepsilon$, we evaluate changes in both variables simultaneously (similar to a grid search). For $\varepsilon$ we test values ranging from $0.1$ to $0.5$, and for $K$ we test values ranging from $2$ to $500$. Results for ROBUSTAD are presented on Figure 6.1. Notice the two green zones where the algorithm works best. One of them, where $K = 2$, makes ROBUSTAD too similar to AD, since there are only 2 intervals. Moreover, there is no learning of $Z$, and higher $\varepsilon$ gives best results because it makes it easier for the extra dimension in the $\theta$-MWU to compensate for a bad choice of $Z$. The other green zone, where our $Z$ learning mechanism really works, is the range of parameters where we are really interested in, and where some learning of $Z$ could happen.

Next, we run a sensitivity test of $\varepsilon$ for algorithm AD. Both AD and ROBUSTAD present guarantees in which the competitiveness decreases with a higher $\varepsilon$, meaning we should set $\varepsilon$ as low as possible, while respecting the algorithm's assumptions. For AD, the assumption is that $\varepsilon \geq \sqrt{\log d/B}$, resulting in $\varepsilon = 0.04$, if we consider $\log$ as $\ln$. As for ROBUSTAD, assumptions are quite stronger:

$$\varepsilon = \sqrt[4]{(\log\log T + \log(1/\varepsilon))\,\frac{\log(d+1)}{B}},$$

which gives $\varepsilon = 0.27$. Results are presented in Figure 6.2. For both algorithms, calibrating $\varepsilon$ according to theory showed to be effective, however, ROBUSTAD's performance still lagged behind. Finally, for the remaining experiments, we will use AD with $\varepsilon = 0.05$ and ROBUSTAD with $\varepsilon = 0.2$ and $K = 100$.

Next, we compare both algorithms with the ones presented by Wang [27].

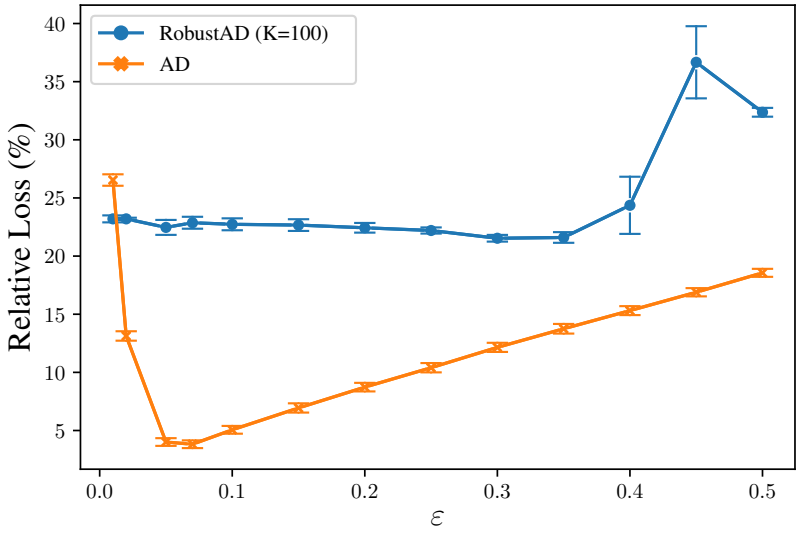


Figure 6.2: Sensitivity to $\varepsilon$ on Wang instances. The intervals represent the standard deviation.

Note that methodologies for obtaining those results differ slightly between our experiments and those of Wang, as explained in the beginning of Section 6.1.2. Particularly, the instances used for AD and ROBUSTAD had a final normalization step (step 5 described in Section 6.1), and the ranges of $\varepsilon$ tested also differ. Results are presented in Figure 6.3. AD and ROBUSTAD's experiments are work of our own, while One-Time Learning and Dynamic Learning's results were reproduced from [27]. We notice the Dynamic Learning algorithm is the one that presents the best experimental results.

***d* sensitivity.** Next, we check the impact of a variation in $d$ in both AD and ROBUSTAD algorithms. In AD, the loss relative to $d$ is of order $O\left(\sqrt[2]{\log d}\right)$, and in ROBUSTAD it is $O\left(\sqrt[4]{\log d}\right)$, so no considerable change is expected. Results can be found on Figure 6.4. Although the impact of varying $d$ seems to be large on ROBUSTAD, in reality, ROBUSTAD is operating in its normal performance range, between relative losses of $20\%$ and $25\%$. Thus, the variation of $d$ had no perceptible impact on any algorithm.

***B* sensitivity.** The next test we present is the sensitivity to $B$. This test is specially important, since we make assumptions of the kind "$B \geq \ldots$" in order to achieve the presented guarantees. The assumption on $B$ for AD is $B \geq \frac{\log(d+1)}{\varepsilon^2}$, so it becomes true when $B \geq 717$. For ROBUSTAD, the assumption is $B \geq \frac{\log d}{\varepsilon^4}(\log\log T + \log(1/\varepsilon))$, and becomes true when $B \geq 1485$. The results are

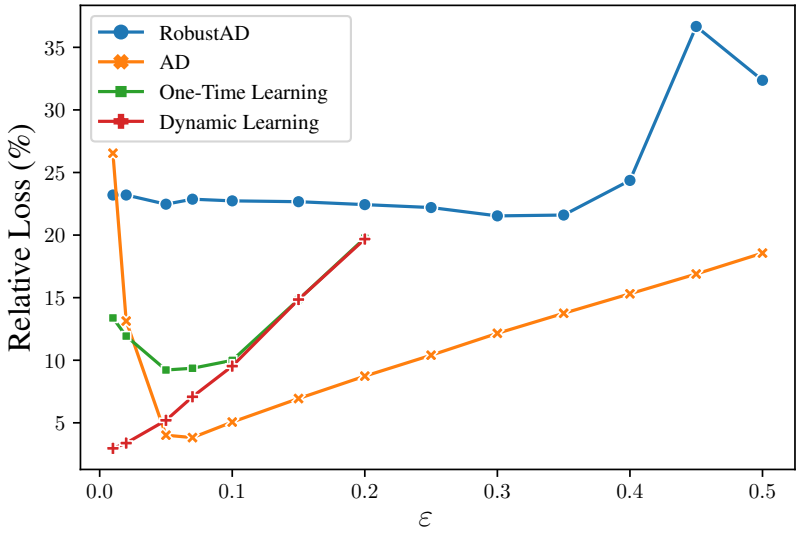


Figure 6.3: Sensitivity to $\varepsilon$ on Wang instances, including Wang's algorithms. Results for One-Time Learning and Dynamic learning were extracted from Wang [27]. Methodologies differ slightly.



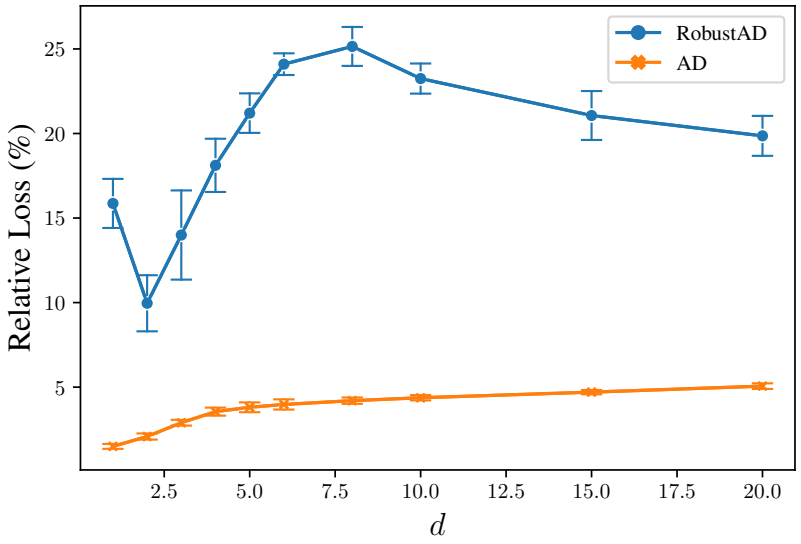


Figure 6.4: Sensitivity to $d$ on Wang instances. The intervals represent the standard deviation.

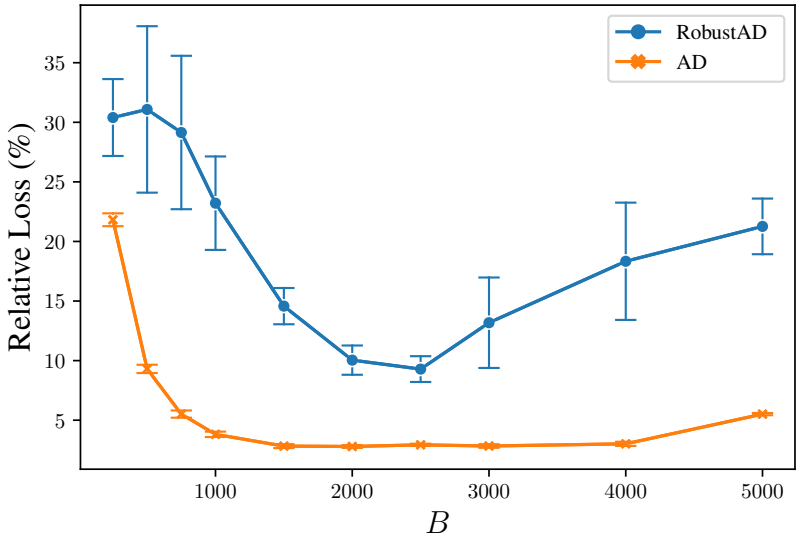


Figure 6.5: Sensitivity to $B$ on Wang instances. The intervals represent the standard deviation.

presented in Figure 6.5. ROBUSTAD appears to be highly sensitive to $B$, so much that he algorithm does not seem suitable for cases when the capacity $B$ is below its theoretical threshold, while AD is much more stable.

***T* sensitivity.** At last, we check the sensitivity on $T$, as presented in Figure 6.6. AD showed to be very robust to variations in $T$, with excellent performance. ROBUSTAD, on the contrary, didn't do so well, and displayed high instability when faced with changes in $T$, specially for the lower values of $T$. A possible reason for the instability could be the role that $T$ plays within the algorithm. Some important parameters have a theoretical dependency on $T$, such as $K$, and the set of $Z$ experts $\mathcal{Z}$,

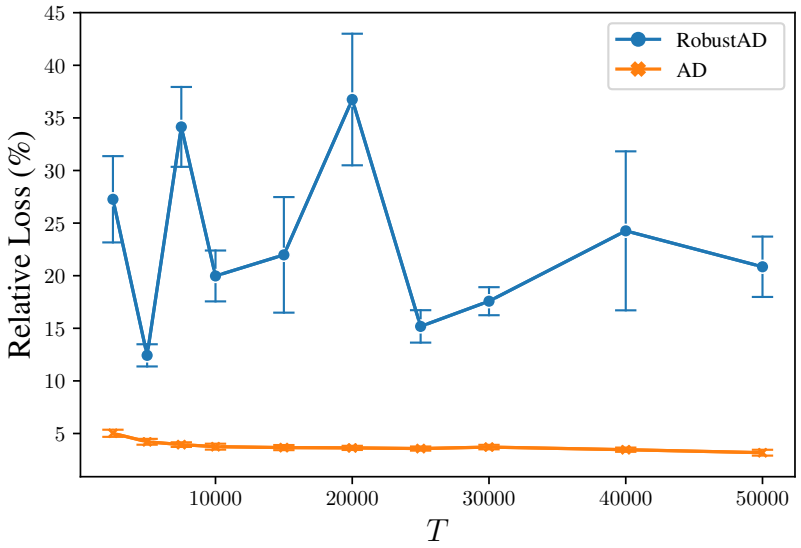


Figure 6.6: Sensitivity to $T$ on Wang instances. The intervals represent the standard deviation.

## 6.2 Chu and Beasley Instances

In the previous section, we only worked with independent and identically distributed (i.i.d) instances, that fit the RANDOMORDER model. However, ROBUSTAD should also work for MIXED model instances, which contain some adversarial parts. In this section, we create more adversarial instances to test both algorithms' robustness. We use Chu and Beasley [10] instances, with some manipulation. We did not use the original instances provided with their paper, but instead we used their generation method to generate instances with $T = 10000$. Also, for every experiment in this section, we run AD with $\varepsilon = 0.05$ and a start phase of size $0.05\,T$, and we run ROBUSTAD with $\varepsilon = 0.2$ and $K = 100$.

Unfortunately, the original Chu and Beasley procedure [10] break some of our model's assumptions, in particular: 1) all occupation entries and rewards must be in the range $[0, 1]$; 2) each constraint must have the same capacity $B$. Thus, some adaptation is needed in order to make those instances compatible with our model. The adapted generation procedure is described below:

- (step 1) Receive $T$, $d$ and $\alpha$ (tightness) as parameters.
- (step 2) Generate all the occupations: $a_{t,j} \leftarrow U(0,1)\ \forall j \in [d],\ \forall t \in [T]$
- (step 3) Since the sum of occupations for each dimension will be similar, if we set $B = \alpha \frac{1}{d} \sum_{j\in[d]} \sum_{t\in[T]} a_{t,j}$, the tightness $\alpha$ will be approximately the same for every dimension.
- (step 4) Generate the rewards for each time step $t$: $c_t = \frac{1}{d} \sum_{j\in[d]} a_{t,j} + 0.5\,U(0,1)$.
- (step 5) If any reward is greater than 1, we normalize the rewards.

Next, we enumerate the changes we made to the original procedure from Chu and Beasley [10]:

1. On step 2, they generated $a_{t,j}$ by selecting a random integer in the range $[0, 1000]$. We scaled that range down to a random real number in the range $[0, 1]$.

2. On step 3, they generated a different capacity for each occupation dimension $j$: $B_j = \alpha \sum_{t\in[T]} a_{t,j}$. We calculated $B$, the average across $B_j$'s, and attributed that same capacity $B$ to every dimension.

3. On step 4, they set the rewards as $c_t = \frac{1}{d} \sum_{j\in[d]} a_{t,j} + 500\,U(0,1)$. We scaled the random part down to $0.5\,U(0,1)$.

**Methodology.** For all remaining experiments, we generate 10 Chu and Beasley instances with $T = 10000$, $d = 5$ and $\alpha = 0.2$, for each configuration, and the presented relative loss is relative to those instances. Notice since the expected sum of occupation for all items in each dimension is $5000$, using $\alpha = 0.2$ would result in an expected capacity of $B = 1000$, similar to the base problem for Wang instances. When manipulation is applied, it is applied on top of those base instances, consequently ruining the tightness $\alpha$, but we do not worry about that.

### 6.2.1 Instances with Exceptional Initial Items

In this section, we simulate an adversary that manipulates only the initial part of the instance, where AD is running its start phase ($0.05\,T$ time steps). Additionally, two types of manipulation will be tested: one where items' quality is improved (higher rewards), and another where it is worsened (lower rewards).

**Higher quality initial items.** In this experiment, we make the first $500$ items better than the rest. This is done by dividing the all rewards by a scale factor $q$, except for the first $500$ rewards. For $q$, we test values $1.25$, $1.5$, $2$, $4$, $8$, $16$. We expect that AD will learn an artificially high value for $Z$, making it algorithm more selective, resulting in only few items being good enough to be picked. Results are presented on Figure 6.7.

The behavior of AD can be explained by two factors. First, since is does not pick any item in the starting phase, the higher the $q$, the higher are the rewards of the items AD is skipping in the start phase. Higher values for $q$ also make increase the initial items' contribution to OPT. Second, the higher the $q$, the more is AD skewed towards judging remaining items as low quality items, thus, not picking them. In fact, most of the times, we checked that AD finished processing the instance without using its full capacity in any of the dimensions.

The behavior observed for ROBUSTAD was not expected. With the current settings, using $K = 100$ intervals in ROBUSTAD and working with instances with $T = 10000$, the $500$ manipulated time steps correspond to 5 manipulated intervals, out of a total of $100$ intervals. Still, the manipulation introduced such a strong bias in the $Z$-MSMW mechanism, that the algorithm needed much more than $5$ new intervals in order to re-establish a feasible value for $Z$. As an example, for $q = 4$ (initial items rewards 4 times higher than the rest), ROBUSTAD needed to run until interval $17$, in average, to re-establish a reasonable value for $Z$. Likewise what happened to AD, ROBUSTAD also did not use its full capacity in any dimension.

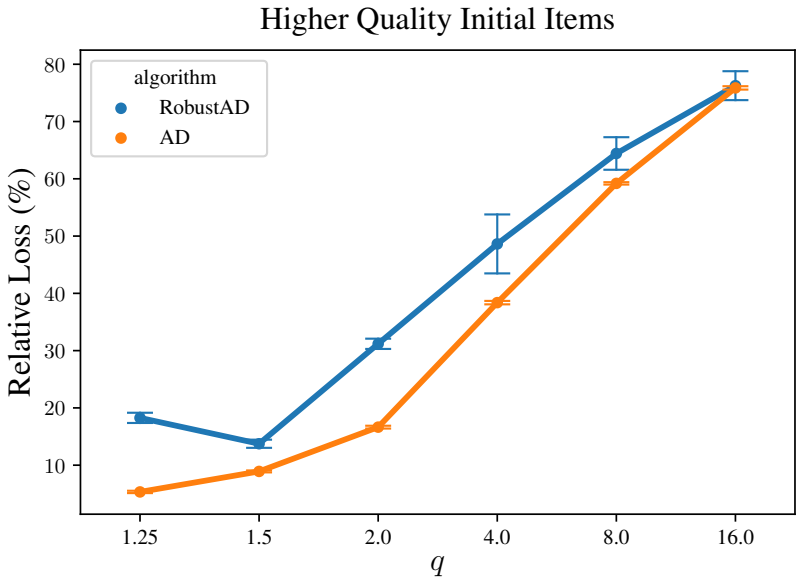


Figure 6.7: Sensitivity to the manipulation magnitude when the initial items have higher quality. The intervals represent the standard deviation.

**Lower quality initial items.** In this experiment, we make the first $500$ items worst than the rest. For that, we divide the initial items' rewards by a scale factor $q$. For $q$, we test values $1.25$, $1.5$, $2$, $4$, $8$ and $16$. With that, we expect AD to become less selective, and to accept lower quality items more frequently. We also expect its performance to be damaged since the capacity could get fully occupied early on, in consequence of the loose selection criterion. Results are available on Figure 6.8.

When the first items have an exceptionally lower quality, a weakness from AD is turned into a strength: skipping the items from the initial phase. With that, it avoids the worst items in the whole instance, which explain the increasing performance until $q = 2$. Afterwards, however, the bias introduced by a maladjusted value of $Z$ dominates the performance of the instance, until it becomes a greedy algorithm, which just picks every items, regardless.

ROBUSTAD, on the other hand, did not seem much impacted by the manipulated instance, displaying its normal performance. However, for $q = 4$, for example, it was able to recover the optimum $Z$ by interval $15$, which still means it operated biased on $15\%$ of the instance. This suggests that, for ROBUSTAD, acting too selective can be more damaging than acting too greedy.

### 6.2.2 Instances with Regime Changes

In this experiment, we manipulate items' rewards based on waves, to see how the algorithms behave when there are changes in the underlying distribution, which

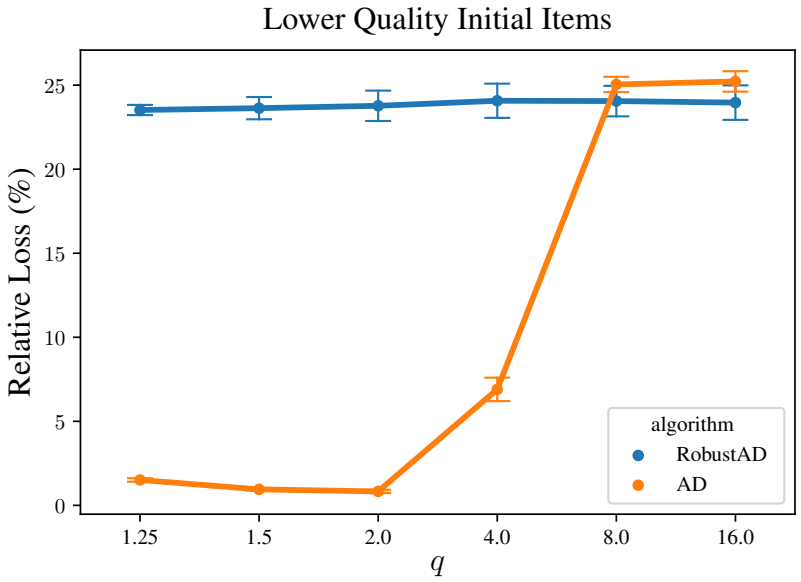


Figure 6.8: Sensitivity to the manipulation magnitude when the initial items have lower quality. The intervals represent the standard deviation.

items' rewards are drawn from. Some real life examples representing this scenario are shocks in the market, resulting in big changes in input costs or sale prices for manufactured goods. We divide the instance into $W$ waves, and we scale down the rewards of each second wave. The scale down is done by a factor $q$, for which we test values $1.25$, $1.5$, $2$, $4$ and $16$. We also experiment with $5$, $10$, $50$, $100$ and $500$ waves. Results are available on Figure 6.9.

Our first observation is that, as the number of waves increases, the performance of both algorithms comes closer to their performance on i.i.d. instances (when no manipulated waves are present). This happens because as the wave lengths get shorter, the algorithm has less time to learn the new wave's pattern and to solidify it as a bias to follow.

We can also see that ROBUSTAD can adapt to new regimes quite fast, given that waves are long enough (when waves $= 5$), especially under high magnitude changes (when $q = 4, 8, 16$). For waves $= 10$, the big loss in performance experienced by ROBUSTAD has an explanation: experiments from previous sections showed that under those instance settings ($T$, $\alpha$, $d$ and $q$), ROBUSTAD could take up to $10$ intervals to adapt to a new value of $Z$. When waves $= 10$, each wave has length $1000$, which is equivalent to 10 intervals for ROBUSTAD, since we are using $K = 100$. That shows that roughly every time ROBUSTAD is able to complete the learning for a new $Z$ value, the regime changes, which is a strongly adversarial scenario.

AD, on the other hand, was not able to adapt well when few regime changes

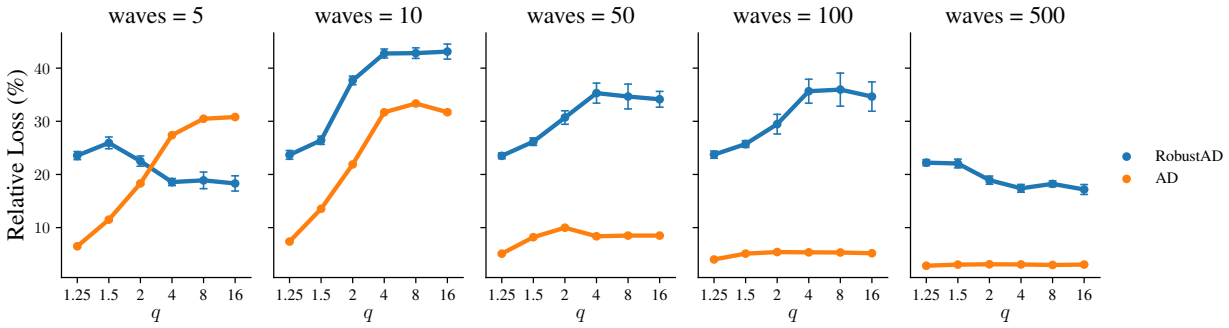


Figure 6.9: Sensitivity to manipulation magnitude when instances are manipulated in waves. The intervals represent the standard deviation.

occur. However, on every other scenario, it still displayed a large advantage in performance over ROBUSTAD. Moreover, it became stable faster, as the number of waves increased. Possibly, this performance is a consequence of using a low $\varepsilon$ (which also appeared to be optimal for AD). That would cause a much slower learning rate, which could hurt the algorithm when there are few regime changes, and help the algorithm become more stable when regime changes are frequent.

### 6.2.3 Instances with Regime Changes in One Occupation Dimension

In experiment, we manipulate instances to introduce regime changes in a single dimension of occupation. In a real-life manufacturing scenario, this would be equivalent to a sudden change in the costs of a single raw material, one of the many inputs needed to manufacture a product. The objective is to evaluate how adaptable are the $\theta$-learning mechanisms within the algorithms. Again, the sequence of time steps $[T]$ is divided into $W$ waves, and each second wave gets manipulated. The manipulation applied is to multiply the first occupation dimension by a factor $q > 1$. We vary $q$ in $1.25$, $1.5$, $2$, $4$, $8$ and $16$, and we vary the number of waves $W$ between $5, 10, 50, 100, 500$. Results are presented on Figure 6.10.

This experiment is especially different from the above, as the previous was designed to test the ability to learn new values for $Z$. A situation where all the occupation dimensions suffer the same manipulation, would be more similar to the previous experiment. This experiment, in contrast, tests the $\theta$-learning mechanisms. In that sense, it appears that ROBUSTAD is not prepared to deal with instances where a single occupation dimension suffers a manipulation. AD, however, showed to be incredibly stable in this case.

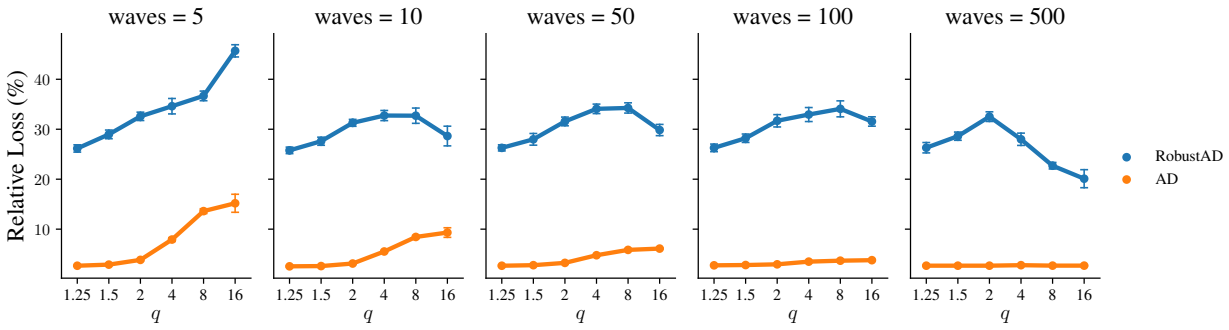


Figure 6.10: Sensitivity to manipulation magnitude when a single occupation dimension is manipulated in waves. The intervals represent the standard deviation.

# 7
# Conclusions and Directions For Future Work

In this work, several contributions were made to the corpus of works directed at solving online packing IPs effectively, the main one being our theoretical analysis. This work was the first, that we are aware of, to consider online packing IPs in the MIXED model. In addition, we presented the first algorithm, to our knowledge, that solves the this problem effectively with provable guarantees.

A second contribution of this work was the experimental section. We proposed modified instances that simulate possible scenarios for the MIXED model. In addition, we implemented two algorithms, AD and ROBUSTAD, and presented results comparing their performance. In almost every experiment, surprisingly, AD displayed superior performance and robustness. In addition, those results show how practical performance may deviate from theoretical guarantees, meaning, algorithms with the similar guarantees can experience very different performances in practice. This highlights the importance of making experiments while selecting an algorithm to implement in a real-life scenario.

Regarding directions for future work, much is still unknown about online packing IPs in the MIXED model. One research direction is to investigate whether assumptions on $B$ can be weakened. Another direction is to inspect how low can $c$ get, where $c$ is the linear coefficient in $\mathsf{Alg} \geq (1 - c\lambda - O(\varepsilon))\mathrm{OPT}_{Stoch}$. One last suggested direction is to study if the adversarial optimum $\mathrm{OPT}_{Adv}$ can also be used in a lower bound for the MIXED model, as in $\mathsf{Alg} \geq \alpha\mathrm{OPT}_{Stoch} + \beta\mathrm{OPT}_{Adv}$.

Lastly, we hope to bring more attention to the MIXED model, in an attempt to encourage the research of algorithms in more generic models, and we hope our analysis techniques will be useful to other future works.

# Bibliography


[1] AGRAWAL, S.; DEVANUR, N. R.. **Fast algorithms for online stochastic convex programming**. In: PROCEEDINGS OF THE TWENTY-SIXTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, SODA 2015, SAN DIEGO, CA, USA, JANUARY 4-6, 2015, p. 1405–1424, 2015.

[2] AGRAWAL, S.; WANG, Z. ; YE, Y.. **A dynamic near-optimal algorithm for online linear programming**. Operations Research, 62(4):876–890, 2014.

[3] ARORA, S.; HAZAN, E. ; KALE, S.. **The multiplicative weights update method: a meta-algorithm and applications**. Theory of Computing, 8(6):121–164, 2012.

[4] BABAIOFF, M.; IMMORLICA, N.; KEMPE, D. ; KLEINBERG, R.. **Online auctions and generalized secretary problems**. SIGecom Exch., 7(2), June 2008.

[5] BANSAL, N.. **On a generalization of iterated and randomized rounding**. In: PROCEEDINGS OF THE 51ST ANNUAL ACM SIGACT SYMPOSIUM ON THEORY OF COMPUTING, STOC 2019, p. 1125–1135, New York, NY, USA, 2019. Association for Computing Machinery.

[6] BÄRMANN, A.; POKUTTA, S. ; SCHNEIDER, O.. **Emulating the expert: Inverse optimization through online learning**. In: PROCEEDINGS OF THE 34TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML 2017, SYDNEY, NSW, AUSTRALIA, 6-11 AUGUST 2017, p. 400–410, 2017.

[7] BEN-TAL, A.; HAZAN, E.; KOREN, T. ; MANNOR, S.. **Oracle-based robust optimization via online learning**. Operations Research, 63(3):628–638, 2015.

[8] BRADAC, D.; GUPTA, A.; SINGLA, S. ; ZUZIC, G.. **Robust algorithms for the secretary problem**. CoRR, abs/1911.07352, 2019.

[9] BUBECK, S.; DEVANUR, N. R.; HUANG, Z. ; NIAZADEH, R.. **Multi-scale online learning: Theory and applications to online auctions and pricing**. Journal of Machine Learning Research, 20(62):1–37, 2019.

[10] CHU, P. C.; BEASLEY, J. E.. **A genetic algorithm for the multidimensional knapsack problem**. Journal of Heuristics, 4(1):63–86, jun 1998.

[11] ESFANDIARI, H.; KORULA, N. ; MIRROKNI, V.. **Allocation with traffic spikes: Mixing adversarial and stochastic models**. ACM Trans. Econ. Comput., 6(3–4), Oct. 2018.

[12] FELDMAN, J.; HENZINGER, M.; KORULA, N.; MIRROKNI, V. S. ; STEIN, C.. **Online stochastic packing applied to display ad allocation**. In: PROCEEDINGS OF THE 18TH ANNUAL EUROPEAN CONFERENCE ON ALGORITHMS: PART I, ESA'10, p. 182–194, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] GARDNER, M.. **Mathematical games**. Scientific American, 202(3):172–186, 1960.

[14] GUPTA, A.; MOLINARO, M.. **How Experts Can Solve LPs Online**, p. 517–529. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[15] HAZAN, E.. **Introduction to online convex optimization**. Foundations and Trends® in Optimization, 2(3-4):157–325, 2016.

[16] HAZAN, E.; KALE, S. ; SHALEV-SHWARTZ, S.. **Near-optimal algorithms for online matrix prediction**. In: Mannor, S.; Srebro, N. ; Williamson, R. C., editors, PROCEEDINGS OF THE 25TH ANNUAL CONFERENCE ON LEARNING THEORY, volumen 23 de **Proceedings of Machine Learning Research**, p. 38.1–38.13, Edinburgh, Scotland, 25–27 Jun 2012. PMLR.

[17] KESSELHEIM, T.; KLEINBERG, R. ; NIAZADEH, R.. **Secretary problems with non-uniform arrival order**. In: PROCEEDINGS OF THE FORTY-SEVENTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '15, p. 879–888, New York, NY, USA, 2015. Association for Computing Machinery.

[18] KESSELHEIM, T.; MOLINARO, M.. **Knapsack Secretary with Bursty Adversary**. In: Czumaj, A.; Dawar, A. ; Merelli, E., editors, 47TH INTERNATIONAL COLLOQUIUM ON AUTOMATA, LANGUAGES, AND PROGRAMMING (ICALP 2020), volumen 168 de **Leibniz International**

**Proceedings in Informatics (LIPIcs)**, p. 72:1–72:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[19] KESSELHEIM, T.; TÖNNIS, A.; RADKE, K. ; VÖCKING, B.. **Primal beats dual on online packing lps in the random-order model**. In: PROCEEDINGS OF THE FORTY-SIXTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '14, p. 303–312, New York, NY, USA, 2014. Association for Computing Machinery.

[20] KLEINBERG, R.. **A multiple-choice secretary algorithm with applications to online auctions**. In: PROCEEDINGS OF THE SIXTEENTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, SODA '05, p. 630–631, USA, 2005. Society for Industrial and Applied Mathematics.

[21] KORULA, N.; MIRROKNI, V. ; ZADIMOGHADDAM, M.. **Online submodular welfare maximization: Greedy beats 1/2 in random order**. In: PROCEEDINGS OF THE FORTY-SEVENTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '15, p. 889–898, New York, NY, USA, 2015. Association for Computing Machinery.

[22] LI, B.; HOI, S. C. H.. **Online portfolio selection: A survey**. ACM Comput. Surv., 46(3):35:1–35:36, 2014.

[23] MEYERSON, A.. **Online facility location**. In: PROCEEDINGS OF THE 42ND IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, FOCS '01, p. 426, USA, 2001. IEEE Computer Society.

[24] MIRROKNI, V. S.; GHARAN, S. O. ; ZADIMOGHADDAM, M.. **Simultaneous approximations for adversarial and stochastic online budgeted allocation**. In: PROCEEDINGS OF THE TWENTY-THIRD ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, SODA '12, p. 1690–1701, USA, 2012. Society for Industrial and Applied Mathematics.

[25] MOLINARO, M.. **Online and random-order load balancing simultaneously**. In: PROCEEDINGS OF THE TWENTY-EIGHTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, SODA 2017, BARCELONA, SPAIN, HOTEL PORTA FIRA, JANUARY 16-19, p. 1638–1650. SIAM, 2017.

[26] MOLINARO, M.; RAVI, R.. **The geometry of online packing linear programs**. Math. Oper. Res., 39(1):46–59, Feb. 2014.

[27] WANG, Z.; YE, Y.; GLYNN, P. W. ; LAI, T. L.. **Dynamic learning mechanisms in revenue management problems**. PhD thesis, Stanford University, 2012.

# A
# Online Learning

## A.1
## MWU Algorithm

This section describes algorithm MWU [3], that solves the Fractional Prediction with Expert Advice problem (Definition 3.1). We reproduce it below:

**Definition A.1** (Fractional Prediction with Expert Advice – Definition 3.1)**.** *Let us define the simplex* $\triangle^d := \{\theta \in \mathbb{R}^d \ : \ \sum_i \theta_i = 1\}$. *In the* Fractional Prediction with Expert Advice *problem, at time* $t$ *the algorithm needs to choose a vector* $\theta_t \in \triangle^d$. *After that, reward vector* $h_t$ *is revealed. The algorithm gets a reward of* $\langle \theta_t, h_t \rangle$. *This process is repeated for* $T$ *iterations. The objective is to maximize the total reward obtained:* $\sum_{t=1}^{T} \langle \theta_t, h_t \rangle$.

Next, we present the algorithm. For vectors $w$, $\theta$ and $h$, we annotate the time step index in subscript, and the dimension index in superscript. The parameter $\varepsilon$ controls the size of each learning step. The algorithm assumes knowledge of a value $\rho$ such that $h_t^i \in [-\rho, \rho]$, for all $t$ and for all $i \in [d]$.

**Algorithm 4** Algorithm MWU

**procedure** MWU($d$, $\rho$, $\varepsilon$)
  Initialize $\theta_1$ and $w_1$ with $1/d$ in each entry of those vectors
  **for** $t = 1, 2, \ldots$ **do**
    Choose/play vector $\theta_t$
    Observe $h_t$
    For each dimension $i$, set $w_{t+1}^i \leftarrow \begin{cases} w_t^i(1+\varepsilon)^{h_t^i/\rho}, & \text{if } h_t^i \geq 0 \\ w_t^i(1-\varepsilon)^{h_t^i/\rho}, & \text{if } h_t^i < 0 \end{cases}$
    $totalWeight \leftarrow \sum_{i=1}^{d} w_{t+1}^i$
    For each dimension $i$, set $\theta_{t+1}^i \leftarrow w_{t+1}^i / totalWeight$

## A.2 Fractional MWU Applied to a Full-Dimensional Simplex

Theorem 5 of [3] proves guarantees of MWU applied to the Fractional problem (Definition 3.1), which occurs in the simplex $\triangle^d$. In this work, we will also need to work with a slightly modified version of the problem where the decision set is the "full-dimensional simplex" that includes the origin, namely the set $\blacktriangle^d := \{\theta \in \mathbb{R}^d \ : \ \sum_i \theta_i \leq 1\}$. In order to use the MWU guarantees (which are defined for the simplex domain), we can reduce the problem in $\blacktriangle^d$ to a problem in $\triangle^{d+1}$, simply by adding a new dimension $d+1$ to account for the extra vertex $0$ in $\blacktriangle^d$. More precisely, we can consider the $(d+1)$-dimensional reward vector $h'_t$ obtained from the original one via $(h'_t)_i = (h_t)_i$ for all $i \in [d]$ and $(h'_t)_{d+1} = 0$.

Applying the MWU to this new problem in $\triangle^{d+1}$, then, gives a solution $\theta'_1, \ldots, \theta'_T \in \mathbb{R}^{d+1}$. Now, if all reward vectors $h'_t \in [-\rho, \rho]^{d+1}$, Theorem 5 of [3] gives us a guarantee that

$$\sum_{t=1}^{T} \langle \theta'_t, h'_t \rangle \geq \sum_{t=1}^{T} \langle \theta'^*, h'_t \rangle - \varepsilon \sum_{t=1}^{T} |\langle \theta'^*, h'_t \rangle| - \frac{\rho \log(d+1)}{\varepsilon}.$$

where $\theta'$ and $\theta'^*$ belong to $\triangle^{d+1}$

Recalling that $(h'_t)_{d+1} = 0$, we know that dimension $d+1$ will not influence our guarantee (even if, at some point, we choose $(\theta'_t)_{d+1} \neq 0$). So, if we define the vectors $\theta_t \in \blacktriangle^d$ as $(\theta_t)_i = (\theta'_t)_i$ for $i \in [d]$, namely, removing the $(d+1)$-th dimension, and we arrive at Lemma 3.2.

## A.3 MSMW Algorithm

This section describes algorithm MSMW [9], that solves the (Integral) Prediction with Expert Advice problem (Definition 3.3). We reproduce it below:

**Definition A.2** (Integral Prediction with Experts – Definition 3.3)**.** *In this problem, there is a set of $n$ "experts". At time $t$, the algorithm needs to choose (possibly randomly) an expert $i_t \in [n]$. After that, it sees the reward $(h_t)_i$ obtained by each expert $i \in [n]$ (so $h_t$ is a reward vector with $n$ coordinates). Again the goal is to maximize the total reward obtained.*

Next, we present the algorithm. For vectors $w$, $p$ and $h$, we annotate the time step index in subscript, and the dimension index in superscript. The parameter $\varepsilon$ controls the size of each learning step. Consider experts with non-negative rewards. Let $c \in \mathbb{R}^n$ be a vector such that $h_t^i \leq c_i$ for every $i \in [n]$ and for all $t$.

**Algorithm 5** Algorithm MSMW

**procedure** MSMW($n$, $\varepsilon$)
  Initialize $p_1$ with $1/n$ in each entry of the vector
  **for** $t = 1, 2, \ldots$ **do**
    Pick an expert $i$ randomly, according to distribution $p_t$
    Observe $h_t$
    For each expert $i$, set $w_{t+1}^i \leftarrow p_t^i \cdot \exp(\varepsilon \cdot {}^{h_t^i}/_{c_i})$
    Find $\lambda^*$ such that $\sum_{i=1}^n w_{t+1}^i / \exp({}^{\lambda^*}/_{c_i}) = 1$
    For each expert $i$, set $p_{t+1}^i \leftarrow w_{t+1}^i / \exp({}^{\lambda^*}/_{c_i})$

# B
# Proof of Lemma 4.3

The ideas, lemmas and proofs in this appendix are a contribution from Marco Molinaro.

## B.1
## Handling Correlations due to Sampling Without Replacement

In order to prove Lemma 4.3, we need to handle correlations due to sampling without replacement. For that we will use the following general lemma.

**Lemma B.1.** *Consider a set of vectors* $\{y^1, \ldots, y^m\} \in [0,1]^d$ *and let* $Y^1, \ldots, Y^k$ *be sampled without replacement from this set. Let* $Z^1, \ldots, Z^k$ *be random vectors in* $\blacktriangle^d$ *such that* $Z^j$ *is a (possibly random) function of* $Y^1, \ldots, Y^{j-1}$ *for all* $j$*. Let* $\tau$ *be a stopping time for the sequence* $((Y^t, Z^t))_t$ *such that* $\tau \leq \frac{m}{2}$*. Then for any* $\varepsilon \in (0, \frac{1}{10}]$ *and* $\delta \in (0,1]$*, with probability at least* $1-\delta$ *we have*

$$\sum_{j \leq \tau} \langle Z^j, Y^j \rangle \;\leq\; (1+4\varepsilon) \sum_{j \leq \tau} \langle \mathbb{E}_{j-1} Z^j, \mathbb{E} Y^j \rangle \;+\; \frac{O(\log d/\delta)}{\varepsilon} \;,$$

*where* $\mathbb{E}_{j-1} Z^j = \mathbb{E}[Z^j \mid (Y_1, Z_1), \ldots, (Y_{j-1}, Z_{j-1})]$.

Special cases of the above lemma have appeared before in the literature, e.g. of [14, Lemma 5]. We will need the following convenient concentration inequality for "martingales with drift".

**Lemma B.2** (Lemma 2.2 of [5])**.** *Let* $X_1, X_2, \ldots, X_k$ *be a sequence of (possibly dependent) random variables with values in* $(-\infty, 1]$ *and such that there is* $\alpha \in (0,1)$ *such that*

$$\mathbb{E}[X_j \mid X_1, \ldots, X_{j-1}] \leq -\alpha \mathbb{E}[X_j^2 \mid X_1, \ldots, X_{j-1}]$$

*for all* $j$*. Then for all* $\lambda \geq 0$

$$\Pr(X_1 + \ldots + X_k > \lambda) \leq e^{-\alpha\lambda}.$$

We also need a maximal Bernstein's inequality for sampling without replacement. It follows by applying Lemma 1 of [14] to the scaled random variables

$\frac{X_i}{M} \in [0,1]$ and using the fact that $\mathrm{Var}(X) \leq \mathbb{E}X$ for every random variable in $[0,1]$ (the last inequality follows from the inequality $\frac{a}{b+c} \geq \min\{\frac{a}{2b}, \frac{a}{2c}\}$, valid for all non-negative reals $a, b, c$).

**Lemma B.3** (Lemma 1 of [14])**.** *Consider a set of real values $x_1, \ldots, x_m$ in $[0, M]$, and let $X_1, \ldots, X_k$ be sampled without replacement from this collection. Assume $k \leq m/2$. Let $S_i = X_1 + \ldots X_i$. Also let $\mu = \frac{1}{m}\sum_i x_i$ and $\sigma^2 = \frac{1}{m}\sum_i (x_i - \mu)^2$. Then for every $\alpha > 0$*

$$\begin{aligned}\Pr\left(\max_{i\leq k} |S_i - i\mu| \geq \alpha\right) &\leq 30\exp\left(-\frac{(\alpha/24)^2}{M(2k\mu + (\alpha/24))}\right) \\ &\leq 30\exp\left(-\min\left\{\frac{(\alpha/24)^2}{4k\mu M}\,,\ \frac{\alpha}{48M}\right\}\right)\end{aligned}$$

Let $\mathcal{F}_j$ be the $\sigma$-algebra generated by $Y^1, \ldots, Y^j$ and $Z^1, \ldots, Z^j$, i.e., the history up to time $j$. We use $\mathbb{E}_{j-1}[\cdot] := \mathbb{E}[\ \cdot\ \mid\ \mathcal{F}_{j-1}]$ to denote expectation conditioned on the history up to time $j-1$.

**Lemma B.4.** *Consider $i \in [d]$. Then with probability at least $1 - \frac{\delta}{d}$ we have $\mathbb{E}_{j-1}Y_i^j \leq (1+2\varepsilon)\mathbb{E}Y_i^j + \frac{O(\log d/\delta)}{m\varepsilon}$ for all $j \leq \frac{m}{2}$ simultaneously.*

*Proof.* Let $\mu = \frac{1}{m}\sum_{j\leq m} y_i^j$, which is the expected value of $Y_i^j$. Moreover, the conditional expectation $\mathbb{E}_{j-1}Y_i^j$ is the average of the $y_i^t$'s that have not appeared up until time $j-1$, namely

$$\mathbb{E}_{j-1}Y_i^j = \frac{\sum_t y_i^t - \sum_{t\leq j-1} Y_i^t}{m-(j-1)} = \frac{m\mu - \sum_{t\leq j-1} Y_i^t}{m-(j-1)}. \tag{B-1}$$

We then bound the last term uniformly for all $j \leq \frac{m}{2}$ using the maximal Bernstein's inequality Lemma B.3.

For that let $\sigma^2 := \frac{1}{m}\sum_t (y_i^t - \mu)^2$ and notice that

$$\sigma^2 = \frac{1}{m}\sum_t (y_i^t)^2 - \mu^2 \leq \frac{1}{m}\sum_t y_i^t = \mu,$$

where the inequality uses $y_i^t \in [0,1]$. Applying Lemma B.3 with

$$\alpha := \varepsilon m\mu + \frac{2\cdot(24)^2}{\varepsilon}(\log d/\delta + \log 30)$$

we get, since $\alpha^2 \geq 4m\mu\,(24)^2\,(\log d/\delta + \log 30)$,

$$\begin{aligned}
&\Pr\left(\max_{j\leq m/2} |\textstyle\sum_{t\leq j} Y_i^j - j\mu| \geq \alpha\right) \\
&\leq 30\exp\left(-\min\left\{\frac{4m\mu(\log d/\delta + \log 30)}{2m\mu}\,,\ \frac{2(24)^2(\log d/\delta + \log 30)/\varepsilon}{48}\right\}\right) \\
&\qquad\qquad \leq 30e^{-(\log d/\delta + \log 30)} \ \leq\ \frac{\delta}{d}.
\end{aligned}$$

Finally, whenever this event holds, Equation (B-1) gives that for all $j \leq m/2$

$$\mathbb{E}_{j-1} Y_i^j \leq \frac{(m-(j-1))\,\mu + \alpha}{m-(j-1)} \leq \mu + \frac{\varepsilon m\mu + O(\frac{\log d/\delta}{\varepsilon})}{m-(j-1)} \leq (1+2\varepsilon)\mu + \frac{O(\log d/\delta)}{m\varepsilon},$$

the last inequality using $j \leq \frac{m}{2}$. This concludes the proof. ■

***Proof of Lemma B.1***. Since $Z^t$ and $Y^j$ are independent conditioned on $\mathcal{F}_{j-1}$, we have

$$\mathbb{E}_{j-1}\langle Z^j, Y^j\rangle \ =\ \langle \mathbb{E}_{j-1} Z^j, \mathbb{E}_{j-1} Y^j\rangle$$

Moreover, applying a union bound on Lemma B.4 over all coordinates $i$, with probability at least $1 - \frac{\delta}{2}$ for all $j \leq \frac{m}{2}$ (in particular for all $j \leq \tau$) we have $\langle \mathbb{E}_{j-1} Z^j, \mathbb{E}_{j-1} Y^j\rangle \leq (1+2\varepsilon)\langle \mathbb{E}_{j-1} Z^j, \mathbb{E} Y^j\rangle + \frac{O(\log d/\delta)}{m\varepsilon}$. Adding over all $j \leq \tau$ we get that, with probability $\geq 1 - \frac{\delta}{2}$,

$$\sum_{j\leq\tau} \mathbb{E}_j\langle Z^j, Y^j\rangle \leq (1+2\varepsilon)\sum_{j\leq\tau}\langle \mathbb{E}_{j-1} Z^j, \mathbb{E} Y^j\rangle + \frac{O(\log d/\delta)}{\varepsilon}.$$

We now show using Lemma B.2 that with good probability the desired quantity $\sum_{j\leq\tau}\langle Z^j, Y^j\rangle$ is close to $\sum_{j\leq\tau}\mathbb{E}_j\langle Z^j, Y^j\rangle$. Define the stopped random variable

$$X_j := \mathbf{1}(\tau \geq j)\cdot\Big[(1-\varepsilon)\langle Z^j, Y^j\rangle - \mathbb{E}_j\langle Z^j, Y^j\rangle\Big].$$

Recall that by definition of stopping time, the event $\mathbf{1}(\tau \geq j)$ is $\mathcal{F}_{j-1}$-measurable, and hence $\mathbb{E}_j X_j = \mathbf{1}(\tau \geq j)\cdot(-\varepsilon\,\mathbb{E}_j\langle Z^j, Y^j\rangle)$. Moreover,

$$\begin{aligned}
\mathbb{E}_j X_j^2 \ &=\ \mathbf{1}(\tau \geq j)\cdot\Big[(1-\varepsilon)^2\,\mathbb{E}_j\underbrace{\langle Z^j, Y^j\rangle^2}_{\leq\langle Z^j, Y^j\rangle} - \underbrace{(2(1-\varepsilon)-1)}_{\geq 0}(\mathbb{E}_j\langle Z^j, Y^j\rangle)^2\Big] \\
&\leq\ \mathbf{1}(\tau \geq j)\cdot\mathbb{E}_j\langle Z^j, Y^j\rangle,
\end{aligned}$$

where the first underbrace is because $\langle Z^j, Y^j\rangle \ \leq\ 1$ and the second because

$\varepsilon \in (0, \frac{1}{2}]$. Together, these observations give

$$\mathbb{E}_j X_j \leq -\varepsilon\, \mathbb{E}_j X_j^2.$$

Then applying Lemma B.2 to the sequence $(X_j)_j$ with $\lambda = \frac{\log 1/2\delta}{\varepsilon}$ we obtain

$$\Pr\left((1-\varepsilon)\sum_{j\leq\tau}\langle Z^j, Y^j\rangle \;>\; \sum_{j\leq\tau}\mathbb{E}_j\langle Z^j, Y^j\rangle + \frac{\log 1/2\delta}{\varepsilon}\right) \;\leq\; \frac{\delta}{2}.$$

Then by union bound with (B.1), with probability at least $1-\delta$ we have

$$\sum_{j\leq\tau}\langle Z^j, Y^j\rangle \;\leq\; \frac{(1+2\varepsilon)}{(1-\varepsilon)}\sum_{j\leq\tau}\langle \mathbb{E}_{j-1} Z^j, \mathbb{E} Y^j\rangle + \frac{O(\log d/\delta)}{\varepsilon}.$$

Verifying that $\frac{(1+2\varepsilon)}{(1-\varepsilon)} \leq 1 + 4\varepsilon$ for all $\varepsilon \in (0, \frac{1}{10}]$ then proves Lemma B.1.

■

## B.2 Proof of Lemma 4.3

Let $\tau_I$ denote the stopping time of AD in an interval $I$. Let $[\tau_I] = \{t \in I : t \leq \tau_I\}$ be the sequence of time steps in $I$ up to $\tau_I$, and let $t_j$ be the time step of the $j^{th}$ random-order item in the interval $I$. Recall that $\boldsymbol{s}_I$ is the number of random-order time steps in $I$ up to $\tau_I$. Then

$$\mathcal{L}_I \;=\; \sum_{t\in[\tau_I]} r_t^* - Z\sum_{t\in[\tau_I]}\langle \theta_t, v_t^* - \tfrac{B\mathbf{1}}{T}\rangle \;=\; \sum_{j\leq \boldsymbol{s}_I} r_{t_j}^* - Z\sum_{j\leq \boldsymbol{s}_I}\langle \theta_{t_j}, v_{t_j}^*\rangle + Z\sum_{t\in[\tau_I]}\langle \theta_t, \tfrac{B\mathbf{1}}{T}\rangle. \tag{B-2}$$

We lower bound the first two terms of the right-hand side with high probability.

By definition we have $\mathrm{OPT}_{Stoch} = \sum_j \mathbb{E}\,[r]_{t_j}^*$. Moreover, since the items are sampled (without replacement), the conditional expected value from time step $t_j$

$$\mu \;:=\; \mathbb{E} r_{t_j}^*$$

is the same for all $(1-\lambda)T$ random-order times $t_j$. Putting these observations together, we get that $\mu = \frac{\mathrm{OPT}_{Stoch}}{(1-\lambda)T}$. Similarly, by feasibility of the optimal solution, $\sum_j v_{t_j}^* \leq B\mathbf{1}$ and the expected occupation vector

$$\vec{\mu} := \mathbb{E} v_{t_j}^*$$

is the same for all random-order times $t_j$, and thus $\vec{\mu} \leq \frac{B\mathbf{1}}{(1-\lambda)T}$.

Moreover, the revenue is concentrated around the expectations: using the maximal Bernstein inequality Lemma B.3 (with $X_j := r^*_{t_j}$, $M = \frac{\mathrm{OPT}_{Stoch}}{B}$, and $\alpha = \varepsilon c\frac{|I|}{T}\mathrm{OPT}_{Stoch}$), we have for a suitably large constant $c$

$$\Pr\left(\max_{\ell\le|I|}\left|\textstyle\sum_{j\le\ell}(r^*_{t_j}-\mu)\right| \ge \varepsilon c\tfrac{|I|}{T}\mathrm{OPT}_{Stoch}\right) \le$$
$$30\exp\left(-\min\left\{\frac{\varepsilon^2\,(\frac{|I|}{T})^2\,\mathrm{OPT}_{Stoch}{}^2}{\frac{|I|}{T}\frac{\mathrm{OPT}_{Stoch}{}^2}{(1-\lambda)B}},\ \frac{\varepsilon\,\frac{|I|}{T}\,\mathrm{OPT}_{Stoch}}{\frac{\mathrm{OPT}_{Stoch}}{B}}\right\}\right)$$
$$\le 30\exp\left(-(1-\lambda)\varepsilon^2\tfrac{|I|}{T}B\right) \le \frac{\delta}{8},$$

where the last inequality uses $B \ge \Omega(\frac{T}{|I|}\frac{\log 1/\delta}{\varepsilon^2(1-\lambda)})$. Notice that since $\boldsymbol{s}_I \le |I|$ this also implies concentration for the sum $\sum_{j\le\boldsymbol{s}_I}(r^*_{t_j}-\mu)$ with random range $j \le \boldsymbol{s}_I$.

In addition, the occupation is also concentrated: applying Lemma B.1, we get

$$\Pr\left(\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},v^*_{t_j}\rangle > (1+4\varepsilon)\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\vec{\mu}\rangle + \Omega\Big(\tfrac{\log d/\delta}{\varepsilon}\Big)\right) \le \frac{\delta}{8}.$$

Notice we can indeed apply Lemma B.1 because $\theta_{t_j}$ is a function of the random-order items before the $j^{th}$ random-order item, namely $(c_{t_1},a_{t_1}),\ldots,(c_{t_{j-1}},a_{t_{j-1}})$ (it also depends on the adversarial items, but these are deterministic), and similarly $\boldsymbol{s}_I$ is a stopping time with respect to the sequence $((c_{t_j},a_{t_j}))_j$ and $\boldsymbol{s}_I \le |I| \le (1-\lambda)T/2$ (the last inequality by assumption). Taking a union bound over both concentration inequalities and using the fact that $B \ge \Omega(\frac{T}{|I|}\frac{\log d/\delta}{\varepsilon^2})$ to upper bound the term $\Omega\Big(\frac{\log d/\delta}{\varepsilon}\Big)$ we get

$$\Pr\left(\sum_{j\le\boldsymbol{s}_I}r^*_{t_j} - Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},v^*_{t_j}\rangle < \boldsymbol{s}_I\mu - (1+4\varepsilon)\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\vec{\mu}\rangle - O\Big(\varepsilon\tfrac{|I|}{T}(\mathrm{OPT}_{Stoch}+ZB)\Big)\right) \le \frac{\delta}{4}.$$

Whenever this event holds, we have

$$\mathcal{L}_I \ge \boldsymbol{s}_I\mu - (1+4\varepsilon)Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\vec{\mu}\rangle + Z\sum_{t\in[\tau_I]}\langle\theta_t,\tfrac{B\mathbf{1}}{T}\rangle - O\Big(\varepsilon\tfrac{|I|}{T}(\mathrm{OPT}_{Stoch}+ZB)\Big). \tag{B-3}$$

By definition of $\vec{\mu}$ the second term is

$$(1+4\varepsilon)Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\vec{\mu}\rangle = (1+4\varepsilon)Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\tfrac{B\mathbf{1}}{(1-\lambda)T}\rangle \tag{B-4}$$

and, since $\boldsymbol{s} \le \tau_I$, the third term can be lower bounded

$$Z\sum_{t\in[\tau_I]}\langle\theta_t,\tfrac{B\mathbf{1}}{T}\rangle \ge Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\tfrac{B\mathbf{1}}{T}\rangle = (1-\lambda)Z\sum_{j\le\boldsymbol{s}_I}\langle\theta_{t_j},\tfrac{B\mathbf{1}}{(1-\lambda)T}\rangle.$$

Applying these observations in (B-3) gives

$$\begin{aligned}\mathcal{L}_I &\geq \boldsymbol{s}_I\mu - (4\varepsilon+\lambda)Z\sum_{j\leq \boldsymbol{s}_I}\langle\theta_{t_j}, \tfrac{B\mathbf{1}}{(1-\lambda)T}\rangle - O\Big(\varepsilon\tfrac{|I|}{T}(\mathrm{OPT}_{Stoch}+ZB)\Big)\\ &\geq \boldsymbol{s}_I\frac{\mathrm{OPT}_{Stoch}}{(1-\lambda)T} - \boldsymbol{s}_I(4\varepsilon+\lambda)\frac{ZB}{(1-\lambda)T} - O\bigg(\varepsilon\frac{|I|}{T}(\mathrm{OPT}_{Stoch}+ZB)\bigg),\end{aligned}$$

where the last inequality uses $\langle\theta_{t_j}, \mathbf{1}\rangle \leq 1$. Since $\boldsymbol{s}_I \leq |I| \leq T$ and we assumed $\lambda \leq \frac{1}{2}$, the second term $\boldsymbol{s}_I(4\varepsilon+\lambda)\frac{ZB}{(1-\lambda)T}$ is at most $O(\varepsilon ZB) + 2\lambda\frac{|I|}{T}ZB$. In total we obtain

$$\mathcal{L}_I \geq \boldsymbol{s}_I\frac{\mathrm{OPT}_{Stoch}}{(1-\lambda)T} - 2\lambda\frac{|I|}{T}ZB - O\bigg(\varepsilon\frac{|I|}{T}(\mathrm{OPT}_{Stoch}+ZB)\bigg),$$

concluding the proof of the lemma.